

# Complessità del calcolo

Dipartimento di Elettronica, Informazione e Bioingegneria  
Politecnico di Milano

## Quanto efficientemente risolviamo un problema?

- Cos'è il *costo* di una computazione?
- Come lo misuriamo?
- Primo passo: costruire strumenti per valutare la complessità di un calcolo
- Secondo passo: analisi di algoritmi e strutture dati notevoli (= risolvono problemi frequenti)
- Obiettivo: saper progettare e combinare algoritmi e strutture dati per realizzare soluzioni *efficienti* (oltre che corrette)

## Quanto efficientemente risolviamo un problema?

- Posto che un dato problema sia risolvibile algoritmicamente, vogliamo sapere quanto ci costa risolverlo
- Effettueremo analisi *quantitative* su:
  - Tempo di calcolo impiegato
  - Spazio occupato (registri, cache, RAM, disco, nastro)
- Analisi fatta su criteri di costo oggettivi (e formalizzabili): non teniamo conto di
  - costi di sviluppo
  - trade-off ambientali tra obiettivi contrastanti...

## Dipendenza dal formalismo di calcolo

- Per la tesi di Church-Turing, un problema è calcolabile o meno indipendentemente dallo strumento usato (purché Turing-completo)
- Possiamo dire lo stesso della complessità del calcolo?
  - Una somma in unario ha efficienza diversa da una in base  $b > 1$
  - Calcolare una traduzione  $y = \tau(x)$  decidendo se  $\exists z \in L_\tau = \{x\$y \mid y = \tau(x)\}$  può essere molto meno efficiente del calcolare la traduzione in qualche caso.
- Non è verosimile assumere che cambiando modello di calcolo non cambi il tempo di esecuzione

# Costruire uno strumento generale

## Costo di calcolo “indipendente” dal formalismo

- Non abbiamo una “tesi di Church-Turing” della complessità
  - ma troveremo una sorta di “tesi di Church-Turing” della complessità
- Costruiamo uno strumento per valutare la complessità temporale e spaziale di un calcolo:
  - che tralasci “considerazioni superflue”
  - utilizzabile per la maggioranza dei modelli di calcolo
- Non avendo un formalismo di calcolo “preferito” per costruire il modello, partiamo dalle TM *deterministiche*

# Complessità temporale e spaziale

Data la computazione  $c_0 \vdash c_1 \vdash c_2 \vdash \dots \vdash c_r$  di  $\mathcal{M}$  (a  $k$  nastri)  
*deterministica*

## Complessità temporale

- La complessità temporale è  $T_{\mathcal{M}}(x) = r$  se  $\mathcal{M}$  termina in  $c_r$ 
  - $\infty$  se non termina
- $\mathcal{M}$  è *deterministica*  $\Rightarrow$  computazione unica sull'ingresso  $x$

## Complessità spaziale

- La complessità spaziale è
  - $S_{\mathcal{M}}(x) = \sum_{j=1}^k \max_{i \in \{0, \dots, r\}} (|\alpha_{ij}|)$ 
    - $\alpha_{ij}$ : contenuto del  $j$ -esimo nastro alla mossa  $i$ -esima
  - Intuitivamente: la somma delle quantità massime di nastro occupate, per ogni nastro
- NB:  $\forall x \quad \frac{S_{\mathcal{M}}(x)}{k} \leq T_{\mathcal{M}}(x)$

# Esempio: riconoscere $L = \{wcu^R\}$ con TM a 1 nastro

## Complessità temporale

- $T_{\mathcal{M}}(x) = |x| + 1$  se  $x \in L$
- $T_{\mathcal{M}}(x) = |\alpha a.ucu^R.b|$  se  $x = \alpha a u c u^R b \beta$
- $T_{\mathcal{M}}(x) = |x| + 1$  se  $x \in \{a, b\}^*$
- $T_{\mathcal{M}}(x) = 2$  se  $x \in c^+.\{a, b\}^*$
- ...

## Complessità spaziale

- $S_{\mathcal{M}}(x) = \left\lfloor \frac{|x|}{2} \right\rfloor + 1$  se  $x \in L$  (includendo  $Z_0$ )
- $S_{\mathcal{M}}(x) = |x| + 1$  se  $x \in \{a, b\}^*$
- ...

# Riducendo all'essenziale

Ci sono un po' troppi dettagli...

## Una prima semplificazione

- Da  $T_{\mathcal{M}}(x)$  e  $S_{\mathcal{M}}(x)$  a  $T_{\mathcal{M}}(n)$  e  $S_{\mathcal{M}}(n)$ , dove  $n$  è la “dimensione” dei dati in ingresso
  - Nel caso precedente  $n = |x|$
  - Esempi pratici: righe/colonne di una matrice, numero di record in una tabella, numero di pacchetti in arrivo dalla rete
- NB: È comunque una semplificazione, perché in generale
  - $|x_1|=|x_2| \not\Rightarrow T_{\mathcal{M}}(x_1)=T_{\mathcal{M}}(x_2)$
  - $|x_1|=|x_2| \not\Rightarrow S_{\mathcal{M}}(x_1)=S_{\mathcal{M}}(x_2)$



# Gestire la variabilità dell'ingresso

## Scelte possibili (sia per $T_{\mathcal{M}}(\cdot)$ che per $S_{\mathcal{M}}(\cdot)$ )

- Caso pessimo:  $T_{\mathcal{M}}(n) = \max_{|x|=n} T_{\mathcal{M}}(x)$
- Caso ottimo:  $T_{\mathcal{M}}(n) = \min_{|x|=n} T_{\mathcal{M}}(x)$
- Caso medio:  $T_{\mathcal{M}}(n) = \frac{\sum_{|x|=n} T_{\mathcal{M}}(x)}{|\mathbf{I}|^n}$  (cioè somma dei tempi per parole lunghe  $n$  / numero di parole lunghe  $n$ )

## Scelte tipiche

- Quasi sempre il caso pessimo
  - È quasi sempre il più rilevante
  - L'analisi risulta più semplice del caso medio (che dovrebbe tenere conto di ipotesi probabilistiche sulla distribuzione dei dati)

## Come crescono $T_{\mathcal{M}}(n)$ e $S_{\mathcal{M}}(n)$ ?

- I valori esatti di  $T_{\mathcal{M}}(n)$  e  $S_{\mathcal{M}}(n)$  per un dato  $n$  non sono particolarmente utili
- Ci interessa il comportamento *asintotico* delle funzioni di costo, ossia quando  $n \rightarrow \infty$ 
  - Non distingueremo quindi  $T_{\mathcal{M}}(n) = n^2 + 13n$  da  $3n^2$ , poiché ambedue hanno comportamenti “simili” a  $n^2$
- Semplificazione aggressiva ( $n^2 \approx 10^{80}n^2$ ) ma molto efficace se usata con raziocinio

# Comportamento asintotico e notazione

## Indicare i tassi di crescita

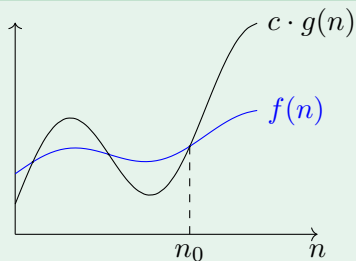
- Introduciamo una notazione per indicare il comportamento asintotico di una funzione:
  - La notazione  $\mathcal{O}$ -grande: limite asintotico superiore
  - La notazione  $\Omega$ -grande: limite asintotico inferiore
  - La notazione  $\Theta$ -grande: limite asintotico sia sup. che inf.
- NB 1: per valori piccoli di  $n$  potrebbe non essere un buon modello
- NB 2: un algoritmo con complessità asintotica maggiore può essere più veloce di uno a complessità minore per valori piccoli di  $n$

# Notazione $\mathcal{O}$ -grande

## Definizione

- Data una funzione  $g(n)$ ,  $\mathcal{O}(g(n))$  è l'insieme  $\mathcal{O}(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 \text{ tali che } \forall n > n_0, 0 \leq f(n) \leq c \cdot g(n)\}$

## Graficamente



- Le funzioni in  $\mathcal{O}(g(n))$  sono dominate da  $c \cdot g(n)$  a partire da  $n_0$

# Esempi

## Facilmente

- $3n^2 + 12n + 35 \in \mathcal{O}(n^2)$
- $5n^3 + 3 \in \mathcal{O}(n^3)$
- $2 \log(n) + \log(\log(n)) \in \mathcal{O}(\log(n))$

## Ma anche

- $3n^2 + 12n + 35 \in \mathcal{O}(n^{20})$
- $5n^3 + 3 \in \mathcal{O}(2^n)$
- $2 \log(n) + \log(\log(n)) \in \mathcal{O}(n)$

## Notazione comune

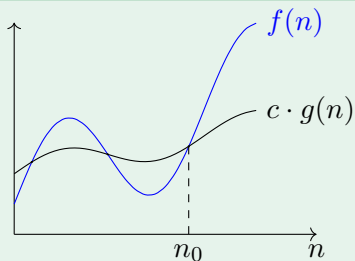
- È comune scrivere  $f(n) = \mathcal{O}(g(n))$  al posto di  $f(n) \in \mathcal{O}(g(n))$  (oppure  $f(n)$  è  $\mathcal{O}(g(n))$ )

# Notazione $\Omega$ -grande

## Definizione

- Data una funzione  $g(n)$ ,  $\Omega(g(n))$  è l'insieme  $\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 \text{ tali che } \forall n > n_0, 0 \leq c \cdot g(n) \leq f(n)\}$

## Graficamente



- Le funzioni in  $\Omega(g(n))$  dominano  $c \cdot g(n)$  a partire da  $n_0$

# Esempi

## Direttamente dalla definizione

- $3n^2 + 12n + 35 \in \Omega(n^2)$
- $7n^2 \log(n) + 15 \in \Omega(n^2 \log(n))$
- $n2^n + n^{50} \in \Omega(n2^n)$

## Di conseguenza, anche

- $3n^2 + 12n + 35 \in \Omega(n)$
- $5n^4 + 3 \in \Omega(\log(n))$ , ma  $5n^4 + 3 \notin \Omega(n^4 \log(n))$
- $n2^n + n^{13} \in \Omega(2^n)$ ,
- $n2^n + n^{13} \in \Omega(n^{70})$

## Proprietà

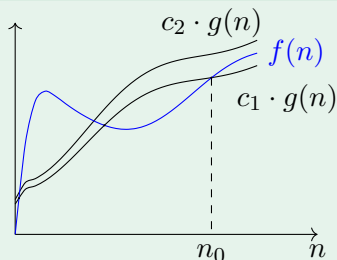
- $f(n) \in \mathcal{O}(g(n))$  se e solo se  $g(n) \in \Omega(f(n))$

# Notazione $\Theta$ -grande

## Definizione

- Data una funzione  $g(n)$ ,  $\Theta(g(n))$  è l'insieme  
 $\Theta(g(n)) = \{f(n) \mid \exists c_1 > 0, c_2 > 0, n_0 > 0 \text{ tali che}$   
 $\forall n > n_0, 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$

## Graficamente



- Le funzioni in  $\Theta(g(n))$  sono comprese tra  $c_1 \cdot g(n)$  e  $c_2 \cdot g(n)$  a partire da  $n_0$



# Esempi

## Direttamente dalla definizione

- $3n^2 + 12n + 35 \in \Theta(n^2)$
- $7n^2 \log(n) + 15 \in \Theta(n^2 \log(n))$
- $n2^n + n^{50} \in \Theta(n2^n)$

## Tuttavia

- $3n^2 + 12n + 35 \notin \Theta(n)$
- $3n^2 + 12n + 35 \notin \Theta(\log(n))$
- $n2^n + n^{50} \notin \Theta(2^n)$
- $n2^n + n^{50} \notin \Theta(n^{100})$

# Proprietà di $\mathcal{O}$ , $\Omega$ , $\Theta$

## Proprietà notevoli

- $f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in \mathcal{O}(g(n)) \wedge f(n) \in \Omega(g(n))$
- Transitività
  - $f(n) \in \Theta(g(n)) \wedge g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$
  - $f(n) \in \mathcal{O}(g(n)) \wedge g(n) \in \mathcal{O}(h(n)) \Rightarrow f(n) \in \mathcal{O}(h(n))$
  - $f(n) \in \Omega(g(n)) \wedge g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$
- Riflessività:
  - $f(n) \in \Theta(f(n))$
  - $f(n) \in \mathcal{O}(f(n))$
  - $f(n) \in \Omega(f(n))$
- Simmetria:  $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$
- Simmetria trasposta:  $f(n) \in \mathcal{O}(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$
- NB:  $\Theta$  è una relazione di equivalenza

# Confronti operativi

## Definizioni come limiti

- Se  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c, c \neq 0, c \neq \infty$  allora  $f(n) \in \Theta(g(n))$ 
  - Gli andamenti asintotici di  $f$  e  $g$  differiscono per una costante moltiplicativa
- Se  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$  allora  $f(n) \in \mathcal{O}(g(n))$ , ma  $f(n) \notin \Theta(g(n))$ 
  - Il comportamento di  $f(n)$  è diverso in modo sostanziale da quello di  $g(n)$  (in particolare,  $g(n)$  cresce più velocemente)
  - Si indica anche, con notazione più sintetica,  $\Theta(f(n)) < \Theta(g(n))$
- L'uso della notazione  $\mathcal{O}, \Omega, \Theta$  permette di evidenziare con facilità la parte più importante di una funzione di complessità

# Una prima applicazione

## Riconoscere $L = \{w c w^r\}$ con TM a 1 nastro

- $T_{\mathcal{M}}(n)$  è  $\Theta(n)$ ,  $S_{\mathcal{M}}(n)$  è anch'essa  $\Theta(n)$
- Ci sono possibilità di miglioramento (= soluzioni a complessità inferiore)?
- Per  $T_{\mathcal{M}}(n)$ , è ben difficile (devo leggere tutta la stringa)
- Per  $S_{\mathcal{M}}(n)$ , sì. Ad esempio, con TM a 2 nastri:
  - Memorizzo solo la posizione  $i$  del carattere da confrontare (in binario)
  - Sposto la testina e confronto i caratteri in posizione  $i$  e  $n - i + 1$
- NB: la complessità spaziale conteggia solamente le celle dei nastri di memoria

# Pseudocodice per TM a 3 nastri $N_a, N_b, N_c$ [complessità tra quadre]

- 1 Ciclo per trovare  $c$ : ad ogni passo incrementa il contatore memorizzato in base 2 in  $N_a$  [ $n \log(n)$ ]
- 2 Ripeti fino a quando il contatore su  $N_a$  è  $= 0$  ( $i$  va da  $n/2$  a 0)
  - 1 Copia il contenuto di  $N_a$  su  $N_b$  [ $\log(n)$ ]
  - 2 Decrementa passo passo  $N_b$  fino a 0 e ad ogni passo sposta verso sx la testina di ingresso [ $i \log(i)$ ]
  - 3 Memorizza il carattere in un nastro  $N_c$  [1]
  - 4 Torna al carattere  $c$  [ $i$ ]
  - 5 Copia il contenuto di  $N_a$  su  $N_b$  [ $\log(n)$ ]
  - 6 Decrementa passo passo  $N_b$  fino a 0 e ad ogni passo sposta verso dx la testina di ingresso di  $N_b$  [ $i \log(i)$ ]
  - 7 Confronta il carattere con quello in  $N_c$ , se diversi halt [1]
  - 8 Decrementa  $N_a$  [ $\log(i)$ ]
  - 9 Torna al carattere  $c$  [ $i$ ]

# Considerazioni sul $T_{\mathcal{M}}(n)$ e $S_{\mathcal{M}}(n)$

## Compromessi tra spazio e tempo

- La nuova soluzione è  $S_{\mathcal{M}}(n) \in \Theta(\log(n))$ , ma  $T_{\mathcal{M}}(n) \in \Theta(n^2 \log(n))$
- È un caso di compromesso spazio-temporale: la soluzione è più lenta ma occupa meno spazio
- Questo esempio giustifica perché in una TM a  $k$  nastri, per convenzione la testina di ingresso si muove in entrambe le direzioni
  - Farla muovere in una sola direzione non cambia il potere computazionale della TM ma impedisce esempi (non banali) di algoritmi con  $S_{\mathcal{M}}(n)$  sub-lineare

## Altri modelli *deterministici* di calcolo

- FSA: hanno sempre  $S_{FSA}(n) \in \Theta(1)$ ,  $T_{FSA}(n) \in \Theta(n)$ 
  - Tecnicamente,  $T_{FSA}(n) = n$ , leggono un carattere per mossa
- PDA: hanno sempre  $S_{PDA}(n) \in \mathcal{O}(n)$ ,  $T_{PDA}(n) \in \Theta(n)$
- TM a nastro singolo?
  - È facile trovare una soluzione  $T_{\mathcal{M}}(n) \in \Theta(n^2)$  per il riconoscimento di  $L = \{wcw^r\}$
  - $S_{\mathcal{M}}(n)$  non potrà mai essere minore di  $\Theta(n)$ 
    - Ecco perché le TM a  $k$  nastri sono il modello principale
  - Non esiste un algoritmo più efficiente di  $T_{\mathcal{M}}(n) \in \Theta(n^2)$  (Dimostrazione complessa. Intuizione: a ogni controllo di ognuna delle  $\frac{n-1}{2}$  coppie di lettere, la TM scandisce una porzione di nastro che passa su  $c$ )
- In generale: TM a nastro singolo sono più potenti dei PDA, ma talvolta meno efficienti!

# I teoremi di accelerazione lineare

## Teorema

*Se  $L$  è accettato da una TM  $\mathcal{M}$  a  $k$  nastri in  $S_{\mathcal{M}}(n)$ , per ogni  $c \in \mathbb{R}^+$  posso costruire una TM  $\mathcal{M}'$  a  $k$  nastri che accetta  $L$  con  $S_{\mathcal{M}'}(n) < c \cdot S_{\mathcal{M}}(n)$*

## Schema della dimostrazione

- Scelgo un fattore di compressione  $r$  tale che  $r \cdot c > 2$
- Per ogni alfabeto  $\Gamma_i$  dell' $i$ -esimo nastro di  $\mathcal{M}$  costruisco  $\Gamma'_i$  di  $\mathcal{M}'$  assegnando un elemento per ogni  $s \in \Gamma_i^r$
- Costruisco l'organo di controllo di  $\mathcal{M}'$  in modo tale per cui:
  - Calcoli con i nuovi simboli sui nastri emulando le mosse di  $\mathcal{M}$  spostando le testine sui nastri ogni  $r$  movimenti di  $\mathcal{M}$
  - Memorizzi la posizione della testina arricchendo ulteriormente gli alfabeti di nastro  $\Gamma_i$  oppure arricchendo l'insieme degli stati



## I teoremi di accelerazione lineare - 2

### Teorema

*Se  $L$  è accettato da una TM  $\mathcal{M}$  a  $k$  nastri in  $S_{\mathcal{M}}(n)$ , posso costruire una TM  $\mathcal{M}'$  a 1 nastro (non nastro singolo) che accetta  $L$  con  $S_{\mathcal{M}'}(n) = S_{\mathcal{M}}(n)$  (concateno i contenuti dei  $k$  nastri su uno solo)*

### Teorema

*Se  $L$  è accettato da una TM  $\mathcal{M}$  a  $k$  nastri in  $S_{\mathcal{M}}(n)$ , per ogni  $c \in \mathbb{R}^+$  posso costruire una TM  $\mathcal{M}'$  a 1 nastro che accetta  $L$  con  $S_{\mathcal{M}'}(n) < c \cdot S_{\mathcal{M}}(n)$  (come sopra + compressione)*

## I teoremi di accelerazione lineare - 3

### Teorema

*Se  $L$  è accettato da una TM  $\mathcal{M}$  a  $k$  nastri in  $T_{\mathcal{M}}(n)$ , per ogni  $c \in \mathbb{R}^+$  posso costruire una TM  $\mathcal{M}'$  a  $k + 1$  nastri che accetta  $L$  con  $T_{\mathcal{M}'}(n) = \max(n + 1, c \cdot T_{\mathcal{M}}(n))$  (come sopra + compressione)*

### Schema di dimostrazione

- Approccio simile alla complessità spaziale: codifichiamo in modo compresso i simboli dell'alfabeto di  $\mathcal{M}$
- Dobbiamo considerare che la compressione è fatta a runtime: minima  $T_{\mathcal{M}'}(n)$  lineare
- Comprimendo  $r$  simboli in uno, nel caso pessimo, possono servirmi 3 mosse di  $\mathcal{M}'$  per emularne  $r+1$  di  $\mathcal{M}$

# Conseguenze pratiche

## L'unico “pasto gratis” è lo speedup lineare

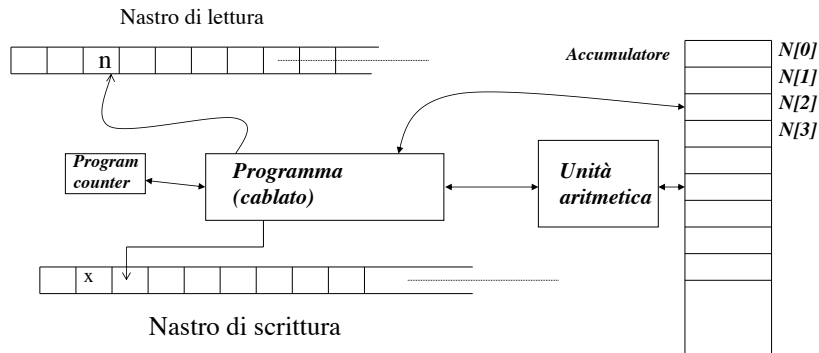
- Lo schema di dimostrazione usato per le TM vale anche per il modello di calcolatore di Von Neumann
  - Equivale a cambiare la dimensione della parola della macchina (ad esempio, da 32 a 64 bit)
- Possiamo avere speedup *lineari* arbitrariamente grandi, aumentando il parallelismo fisico (stanti i limiti della termodinamica/trasmissione dei segnali)
- Miglioramenti più che lineari nel tempo di calcolo possono essere ottenuti solo *cambiando algoritmo*
  - Concepire/utilizzare algoritmi efficienti è di gran lunga più efficace della forza bruta

# Modelli di calcolo a confronto

## TM vs. calcolatori reali

- Differenze “marginali”: un calcolatore è in grado di fare operazioni aritmetiche su tipi a dimensione finita in tempo costante (e.g., add in un ciclo di clock), mentre la TM richiede di propagare gli effetti al singolo bit uno per uno
  - Il calcolatore opera con un alfabeto molto vasto,  $|\mathbf{I}| = 2^w$ , dove  $w$  è la sua dimensione di parola
- Un calcolatore può accedere direttamente ad una cella di memoria, una TM impiega  $\Theta(n)$  dove  $n$  è la distanza della stessa dalla posizione della testina
- Cambiamo modello di calcolo, avvicinandoci ai calcolatori reali

# La macchina RAM



# La macchina RAM

## Un calcolatore semplificato

- La macchina RAM è dotata di un nastro di lettura  $In$  e uno di scrittura  $Out$  come la TM
- Assumiamo che il programma sia cablato nell'organo di controllo, così come la logica del program counter
- La RAM è dotata di una memoria con accesso a indirizzamento diretto  $N[n]$ ,  $n \in \mathbb{N}$  al posto dei nastri di memoria: l'accesso non necessita di scorrimento delle celle
- Le istruzioni di un programma usano normalmente come sorgente il primo operando e come destinazione  $N[0]$
- Ogni cella contiene *un intero*

# La macchina RAM

## Instruction set e semantica pseudo-RTL

| Istruzione | Semantica                          | Istruzione | Semantica                              |
|------------|------------------------------------|------------|--|
| LOAD X     | $N[0] \leftarrow N[X]$             | READ X     | $N[X] \leftarrow \text{In}$            |
| LOAD= X    | $N[0] \leftarrow X$                | READ* X    | $N[N[X]] \leftarrow \text{In}$         |
| LOAD* X    | $N[0] \leftarrow N[N[X]]$          | WRITE X    | $\text{Out} \leftarrow N[X]$           |
| STORE X    | $N[X] \leftarrow N[0]$             | WRITE= X   | $\text{Out} \leftarrow X$              |
| STORE* X   | $N[N[X]] \leftarrow N[0]$          | WRITE* X   | $\text{Out} \leftarrow N[N[X]]$        |
| ADD X      | $N[0] \leftarrow N[0] + N[X]$      | JUMP l     | $\text{PC} \leftarrow l$               |
| SUB X      | $N[0] \leftarrow N[0] - N[X]$      | JZ l       | $\text{PC} \leftarrow l$ se $N[0] = 0$ |
| MUL X      | $N[0] \leftarrow N[0] \times N[X]$ | JGZ l      | $\text{PC} \leftarrow l$ se $N[0] > 0$ |
| DIV X      | $N[0] \leftarrow N[0]/N[X]$        | JLZ l      | $\text{PC} \leftarrow l$ se $N[0] < 0$ |
| HALT       | —                                  |            |  |

# Test (semplice) di primalità in assembly RAM

Funzione  $\text{isPrime}(n) = \text{if } n \text{ is prime then } 1 \text{ else } 0$

|    |               |  |
|----|---------------|--|
| 1  | READ 1        | il valore di ingresso $n$ è memorizzato in $N[1]$      |
| 2  | LOAD= 1       | Se $n = 1$ , esso banalmente non è primo               |
| 3  | SUB 1         |  |
| 4  | JZ no         |  |
| 5  | LOAD= 2       | $N[2]$ è inizializzato a 2                             |
| 6  | STORE 2       |  |
| 7  | loop: LOAD 1  | Se $N[1] = N[2]$ allora $n$ è primo                    |
| 8  | SUB 2         |  |
| 9  | JZ yes        |  |
| 10 | LOAD 1        | Se $N[1] = (N[1] \text{ DIV } N[2]) \cdot N[2]$ allora |
| 11 | DIV 2         | $N[2]$ è un divisore di $N[1]$ ;                       |
| 12 | MUL 2         | quindi $N[1]$ non è primo                              |
| 13 | SUB 1         |  |
| 14 | JZ no         |  |
| 15 | LOAD 2        | $N[2]$ è incrementato di 1 e il ciclo viene ripetuto   |
| 16 | ADD= 1        |  |
| 17 | STORE 2       |  |
| 18 | JUMP loop     |  |
| 19 | yes: WRITE= 1 |  |
| 20 | HALT          |  |
| 21 | no: WRITE= 0  |  |
| 22 | HALT          |  |



# Complessità del precedente programma

## Analisi temporale

- Assunzione di base: istruzioni singole a costo costante  $c_i$ , dove  $i$  è l'indice della riga
- Le istruzioni 1 – 6 sono eseguite al più una volta  $\rightarrow$  costo  $= c_1 + c_2 + c_3 + c_4 + c_5 + c_6$ , è una costante  $k_1$
- Analogamente per le istruzioni 19 – 22, costo costante  $k_3$
- Le istruzioni 7 – 18 hanno costo costante  $k_2$ , ma sono eseguite, nel caso pessimo  $n$  volte
- $T_{RAM}(n) = k_1 + nk_2 + k_3 = \Theta(n)$
- $S_{RAM}(n) = 3 = \Theta(1)$  (uso solo 3 celle di mem)
- N.B.: questa volta  $n$  non è la lunghezza dell'ingresso!

## Analisi di altri algoritmi

### Riconoscere $L = \{wcw^R\}$

- $T_{RAM}(n) = \Theta(n)$
- $S_{RAM}(n) = \Theta(n)$

### Ricerca Binaria

- Input: una sequenza ordinata di interi, ed un numero da cercare in essa
- Output: 1 se l'elemento cercato esiste nella sequenza, 0 altrimenti

# Ricerca Binaria: codice (parte 1)

Assunzioni: sequenza già caricata in memoria

- $N[1]$  = indirizzo della prima cella della sequenza;  $N[2]$  = numero di elementi; quindi la sequenza va dalla cella  $N[N[1]]$  alla cella  $N[N[1] + N[2] - 1]$

|    |              |   |
|----|--------------|---|
| 1  | READ 3       | L'elemento da cercare è letto e memorizzato in $N[3]$     |
| 2  | LOAD 1       |   |
| 3  | STORE 4      | $N[4]$ è inizializzato con l'indirizzo del primo numero   |
| 4  | ADD 2        |   |
| 5  | SUB= 1       |   |
| 6  | STORE 5      | $N[5]$ è inizializzato con l'indirizzo dell'ultimo numero |
| 7  | loop: LOAD 5 | Se l'indirizzo di $N[5]$ precede quello di $N[4]$ siamo   |
| 8  | SUB 4        | giunti ad avere una porzione di sequenza che è vuota,     |
| 9  | JLZ no       | quindi l'elemento cercato non esiste                      |
| 10 | LOAD 5       |   |
| 11 | ADD 4        |   |
| 12 | DIV= 2       |   |
| 13 | STORE 6      | $N[6]$ contiene ora l'indirizzo dell'elemento centrale    |
| 14 | LOAD* 6      |   |
| 15 | SUB 3        |   |

## Ricerca Binaria: codice (parte 2)

|    |      |           |   |
|----|------|-----------|---|
| 16 | JZ   | yes       | Se $N[3] = N[N[6]]$ l'elemento cercato esiste |
| 17 | JGZ  | fst       | Se $N[3] < N[N[6]]$ cerca nella prima metà    |
| 18 | JUMP | snd       | Se $N[3] > N[N[6]]$ cerca nella seconda metà  |
| 19 | fst: | LOAD 6    |   |
| 20 |      | SUB= 1    |   |
| 21 |      | STORE 5   | Memorizza $N[6] - 1$ in $N[5]$                |
| 22 |      | JUMP loop |   |
| 23 | snd: | LOAD 6    |   |
| 24 |      | ADD= 1    |   |
| 25 |      | STORE 4   | Memorizza $N[6] + 1$ in $N[4]$                |
| 26 |      | JUMP loop |   |
| 27 | yes: | WRITE= 1  |   |
| 28 |      | HALT      |   |
| 29 | no:  | WRITE= 0  |   |
| 30 |      | HALT      |   |

Il ciclo viene eseguito al più  $\log_2(n)$  volte, per una sequenza di  $n$  elementi, in quanto ogni volta la dimensione dell'array viene dimezzata. Costo complessivo:

$$T_{RAM}(n) = \Theta(\log(n)) < \Theta(n) \leq T_{TM}(n)$$

## Numeri molto grandi

- Consideriamo il caso del calcolo di  $2^{2^n}$  con una RAM
- Uno schema di implementazione possibile è

```
read(n);  
x=2;  
for(int i=0; i< n; i++) x = x*x;  
write(x);
```
- Che complessità temporale ha l'implementazione qui sopra?  
 $T_{RAM}(n) = k_1 + k_2 + nk_3 + k_4 = \Theta(n)$
- Qualcosa non va: mi servono  $2^n$  bit solo per scrivere il risultato!

# Un criterio di costo più preciso

## Quando contare i singoli bit

- Il criterio di costo precedente considera un intero arbitrario di dimensione costante
  - È una semplificazione: “1” è più corto di “1000000”
- L'approssimazione regge fin quando una singola parola della macchina reale contiene gli interi che maneggiamo
- Se questo non accade, dobbiamo tenere conto del numero di cifre necessarie per rappresentare un intero
  - Caricare e salvare interi non è più a costo costante
  - Le operazioni elementari (somma, prodotto...) neppure

# Criterio di costo logaritmico

## Quando contare i singoli bit

- Copiare/spostare/scrivere/leggere un intero  $i$  costa tanto quanto il suo numero di cifre in base  $b$ :  $\log_b(i) = \Theta(\log(i))$ 
  - Con  $b = 2$  il costo è il numero di bit usati per rappresentare  $i$
- Il costo delle operazioni aritmetico/logiche elementari dipende dall'operazione (definiamo  $d = \log_2(i)$ )
  - Addizioni, sottrazioni, op. al bit  $\Theta(d)$
  - Moltiplicazioni e divisioni: metodo scolastico  $\Theta(d^2)$ 
    - ma esistono algoritmi molto più efficienti
- I salti e la HALT sono a costo costante

# Tabella dei costi logaritmici

|          |   |
|----------|---|
| LOAD= x  | $\log(x)$   |
| LOAD x   | $\log(x) + \log(N[x])$                                      |
| LOAD* x  | $\log(x) + \log(N[x]) + \log(N[N[x]])$                      |
| STORE x  | $\log(x) + \log(N[0])$                                      |
| STORE* x | $\log(x) + \log(N[x]) + \log(N[0])$                         |
| ADD= x   | $\log(N[0]) + \log(x)$                                      |
| ADD x    | $\log(N[0]) + \log(x) + \log(N[x])$                         |
| ADD* x   | $\log(N[0]) + \log(x) + \log(N[x]) + \log(N[N[x]])$         |
| ...      |   |
| READ x   | $\log(\text{valore input corrente}) + \log(x)$              |
| READ* x  | $\log(\text{valore input corrente}) + \log(x) + \log(N[x])$ |
| WRITE= x | $\log(x)$   |
| WRITE x  | $\log(x) + \log(N[x])$                                      |
| WRITE* x | $\log(x) + \log(N[x]) + \log(N[N[x]])$                      |
| JUMP     | 1   |
| JGZ      | $\log(N[0])$  |
| JZ       | $\log(N[0])$  |
| HALT     | 1   |



# Test di primalità con criterio logaritmico

## Solo i punti essenziali, MUL e DIV scolastiche

|    |       |           |   |
|----|-------|-----------|---|
| 7  | loop: | LOAD 1    | $\log(1) + \log(n)$   |
| 8  |       | SUB 2     | $\log(n) + \log(2) + \log(N[2])$                            |
| 9  |       | JZ yes    | $\log(N[0])$  |
| 10 |       | LOAD 1    | $\log(1) + \log(n)$   |
| 11 |       | DIV 2     | $\log((n)^2) + \log(2) + \log((N[2]))^2$                    |
| 12 |       | MUL 2     | $\log((n/N[2]))^2 + \log(2) + \log((N[2]))^2 < (\log(n))^2$ |
| 13 |       | SUB 1     | $\log(N[0]) + \log(1) + \log(n) < 2 \log(n)$                |
| 14 |       | JZ no     | $\leq \log(n)$  |
| 15 |       | LOAD 2    | $\leq \log(n) + k$  |
| 16 |       | ADD= 1    | ...   |
| 17 |       | STORE 2   |   |
| 18 |       | JUMP loop |   |

- Si può facilmente maggiorare la singola iterazione del ciclo con  $\Theta((\log(n))^2)$
- Ergo la complessità temporale complessiva è  $\Theta(n(\log(n))^2)$

# Calcolo di $2^{2^n}$ , complessità a costo logaritmico

Analisi di caso pessimo, MUL scolastica, indicati solo costi prevalenti

|    |              |  |
|----|--------------|--|
| 1  | READ 2       | $\log(n)$  |
| 2  | LOAD= 2      | $\log(2) = k$                                    |
| 3  | STORE 1      | $\log(2) = k$                                    |
| 4  | loop: LOAD 1 | $\log(2^{2^{n-1}}) = 2^{n-1}$                    |
| 5  | MUL 1        | $(\log(2^{2^{n-1}}))^2 = (2^{n-1})^2 = 2^{2n-2}$ |
| 6  | STORE 1      | $\log(2^{2^n}) = 2^n$                            |
| 7  | LOAD 2       | $\log(n)$  |
| 8  | SUB= 1       | $\log(n)$  |
| 9  | STORE 2      | $\log(n-1) \leq \log(n)$                         |
| 10 | JGZ loop     | $\log(n-1) \leq \log(n)$                         |
| 11 | WRITE 1      | $\log(2^{2^n}) = 2^n$                            |
| 12 | HALT         | 1  |

$$T_{RAM}(n) = \mathcal{O}(\log(n) + n(2^{n-1} + 2^{2n-2} + 2^n + 3 \log(n)) + 2^n) = \mathcal{O}(n2^{2n-2})$$

...possiamo scriverlo meglio?

# Rapporti tra criteri di costo

## Rianalizzando

- Riconoscere  $L = \{w c w^R\}$  è  $\Theta(n \log(n))$  (colpa del contatore)
  - Peggio della macchina di Turing! (Avevo  $\Theta(n)$ )
  - e con la RAM non si può fare di meglio!
- Ricerca binaria:  $\Theta(\log(n)^2)$  (colpa degli indici)

## Alcune ipotesi

- Che relazione intercorre tra il costo a criterio di costo costante ( $c_{cost}$ ) e il costo a criterio di costo logaritmico ( $c_{log}$ )?
  - $c_{log} = c_{cost} \cdot \log(n)$ ?
  - $c_{log} = c_{cost} \cdot \log(c_{cost})$ ?
- Spesso ma non sempre:
  - Per il calcolo di  $2^{2^n}$  abbiamo un  $c_{log} \geq 2^n$ 
    - infatti complessità temporale  $\geq$  complessità spaziale, che qui è  $\Theta(2^n)$

# Scelta del criterio di costo

## Quale criterio scegliere?

- Occorre utilizzare il buon senso!
- Se
  - l'elaborazione non altera l'ordine di grandezza dei dati di ingresso
  - la memoria allocata inizialmente (staticamente?) può non variare a run-time
  - quindi non dipende dai dati
  - e quindi una singola cella è considerabile elementare e con essa le operazioni relative
  - cioè la dimensione di ogni singolo elemento in ingresso non varia significativamente nell'esecuzione dell'algoritmo (= stesso numero di cifre per tutta l'esecuzione)

allora va bene il criterio di costo costante

- Altrimenti (fattoriale,  $2^{2^n}$ , ...) è indispensabile il criterio di costo logaritmico: l'unico "garantito"!

# Rapporti tra complessità con diversi modelli di calcolo

## Modelli di calcolo diversi → diversa efficienza

- Risolvere lo stesso problema con macchine diverse può dare luogo a complessità diverse
  - Ricerca binaria → accesso diretto
  - Riconoscimento  $wcw^R$  → accesso e memorizzazione sequenziale
- Non esiste un modello migliore in assoluto
- Non esiste nemmeno un analogo della tesi di Church per la complessità, però...
- E' possibile stabilire almeno una relazione – di maggiorazione – a priori tra le complessità di diversi modelli di calcolo.

# “Teorema” (tesi) di correlazione polinomiale

(in analogia con la tesi di Church)

- Sotto “ragionevoli” ipotesi di criteri di costo,
  - NB Il criterio di costo costante per la RAM non è “ragionevole” in assoluto!

se un problema è risolvibile mediante il modello  $\mathcal{M}_1$  con complessità (spaziale o temporale)  $C_1(n)$ , allora è risolvibile da un qualsiasi altro modello (Turing-completo)  $\mathcal{M}_2$  con complessità  $C_2(n) \leq \pi(C_1(n))$ , dove  $\pi(\cdot)$  è un opportuno polinomio

- E' una sorta di analogo della tesi di Church-Turing per la complessità, che stabilisce una relazione (di maggiorazione) a priori tra diversi modelli di calcolo
- Dimostriamo il teorema (non più *tesi!*) di correlazione (temporale) polinomiale tra TM e RAM

# Correlazione temporale tra TM a $k$ nastri e RAM

## RAM simula TM a $k$ nastri: simulazione delle azioni

- Mappiamo la TM sulla RAM:
  - Stato della TM  $\rightarrow$  Prima cella di memoria della RAM
  - Una cella RAM per ogni cella del nastro
  - Suddividiamo la restante memoria della RAM in blocchi da  $k$  celle
- Riempiamo i blocchi con questa strategia:
  - Blocco 0: posizione delle  $k$  testine
  - Blocco  $n$ ,  $n > 0$ :  $n$ -esimo simbolo di ognuno dei  $k$  nastri
- La RAM emula la lettura di un carattere sotto la testina con un accesso indiretto, usando l'indice contenuto nel blocco 0

## Correlazione temporale tra TM a $k$ nastri e RAM

### RAM simula TM a $k$ nastri: lettura

- Lettura del blocco 0 e dello stato ( $\Theta(k)$  mosse)
- Lettura dei valori sui nastri in corrispondenza delle testine ( $\Theta(k)$  accessi indiretti)

### RAM simula TM a $k$ nastri: Scrittura

- Scrittura dello stato ( $\Theta(1)$ )
- Scrittura delle celle dei nastri ( $\Theta(k)$  accessi indiretti)
- Scrittura nel blocco 0 per aggiornare le posizioni delle  $k$  testine ( $\Theta(k)$ )

### Complessità dell'emulazione

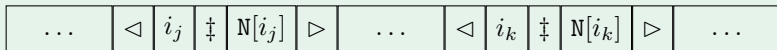
- RAM emula una mossa della TM con  $\Theta(k)$  mosse:
  - $T_{RAM}(n) = \Theta(T_M(n))$  (costo costante)
  - $T_{RAM}(n) = \Theta(T_M(n) \log(T_M(n)))$  (costo logaritmico)
    - un accesso indiretto a  $i$  costa  $\log(i)$



# Correlazione temporale tra TM a $k$ nastri e RAM

## TM a $k$ nastri simula RAM (senza MUL/DIV per semplicità)

- Organizziamo un nastro della TM così:



- Il nastro è inizialmente vuoto: salviamo solo le celle in cui è avvenuta una STORE
- $i_j$  e  $N[i_j]$  sono rappresentati in codifica binaria
- Usiamo un ulteriore nastro per contenere  $N[0]$  in binario
- Usiamo un ultimo nastro (di *servizio*) per consentirci di spostare dati quando dobbiamo salvare per la prima volta  $N[i_j]$  ma  $N[i_k]$  e  $N[i_l]$ ,  $i_k < i_j < i_l$ , sono state già salvate

# Correlazione temporale tra TM a $k$ nastri e RAM

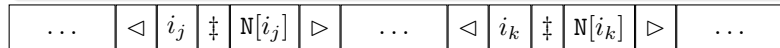
## TM a $k$ nastri simula RAM: simulare istruzioni

- **LOAD  $x$ :** cerco  $x$  sul nastro principale, copio la porzione accanto nella zona dati di  $N[0]$  usando il nastro di supporto
- **STORE  $x$ :** cerco  $x$  sul nastro principale:
  - Se lo trovo, salvo il valore di  $N[0]$  in  $N[x]$ 
    - questo può richiedere l'uso del nastro di servizio se il numero di celle già occupate non è uguale
  - Altrimenti, creo dello spazio usando il nastro di servizio se necessario e salvo
- **ADD  $x$ :** cerco  $x$ , copio  $N[x]$  sul nastro di supporto, calcolo la somma scrivendo direttamente in  $N[0]$
- In generale: simulare una mossa della RAM richiede alla TM un numero di mosse  $\leq c \cdot$  (lunghezza del nastro principale), con  $c$  costante opportuna

# Correlazione temporale tra TM a $k$ nastri e RAM

## Lemma (Occupazione sul nastro principale)

Lo spazio occupato sul nastro principale è  $\mathcal{O}(T_{RAM}(n))$



## Dimostrazione.

- Ogni cella  $i_j$ -esima della RAM occupa  $\log(i_j) + \log(N[i_j]) + 3$
- Ogni cella della RAM viene materializzata solo se la RAM effettua una STORE
- La STORE costa alla RAM  $\log(i_j) + \log(N[i_j])$
- Per riempire  $r$  celle la RAM impiega un tempo ( $\leq T_{RAM}(n)$ ) almeno proporzionale a  $\sum_{j=1}^r \log(i_j) + \log(N[i_j])$ 
  - come lo spazio che occupano sul nastro della TM

# Correlazione temporale tra TM a $k$ nastri e RAM

## Concludendo

- La TM impiega al più  $\Theta(T_{RAM}(n))$  per simulare una mossa della RAM
- Se la RAM ha complessità  $T_{RAM}(n)$  essa effettua al più  $T_{RAM}(n)$  mosse (ogni mossa costa almeno 1)
  - a costo costante sono esattamente  $T_{RAM}(n)$ , a costo logaritmico sono di meno
- La simulazione completa della RAM da parte della TM costa al più  $\Theta((T_{RAM}(n))^2)$ ; il legame tra  $T_{RAM}(n)$  e  $T_{TM}(n)$  è polinomiale

# Impatto della tesi di correlazione polinomiale

## Osservazioni

- E' vero che i polinomi possono anche essere  $n^{1000}$ , ma è sempre meglio dell' "abisso" esponenziale ( $n^c$  contro  $2^n$ )
- Grazie al teorema di correlazione polinomiale possiamo parlare della classe dei problemi risolvibili in tempo/spazio polinomiale (non di quelli risolvibili in tempo quadratico!)
  - la classe non dipende dal modello adottato
- Si è quindi da tempo adottata l'analogia:
  - classe dei problemi "trattabili" in pratica = classe dei problemi risolvibili in tempo polinomiale (P)
  - La teoria include in P anche i problemi con complessità  $n^{1000}$  (sempre meglio di quelli a complessità esponenziale)
  - in pratica i problemi di interesse applicativo (ricerche, cammini, ottimizzazioni, ...) che sono in P hanno anche grado del polinomio accettabile

## e avvertimenti

- Attenzione al parametro di dimensione dei dati:
  - lunghezza della stringa di ingresso (valore assoluto)
  - valore del dato ( $n$ )
  - numero di elementi di una tabella, di nodi di un grafo, di righe di una matrice, ...
  - ...
  - tra tali valori sussistono certe relazioni, ma non sempre esse sono lineari (il numero  $n$  richiede una stringa di ingresso di lunghezza  $\log(n)$ )
- La ricerca binaria implementata con una TM viola il teorema di correlazione polinomiale??
  - Attenzione all'ipotesi: riconoscimento di linguaggio  $\rightarrow$  dati non già in memoria  $\rightarrow$  complessità almeno lineare