La complessità del calcolo

- In questo corso:
  - Cominciamo con un esame critico del problema e dell'approccio
    - cosa è il "costo" di una computazione
    - come lo misuriamo
    - …
  - Andiamo alla ricerca di principi di validità generale
  - Costruiamo una capacità di inquadramento nel giusto ambito di singoli problemi
  - Passiamo quindi alla analisi di algoritmi notevoli
  - Studiamo l'efficienza delle strutture dati
  - Chiudiamo con alcuni argomenti di sapore più avanzato
    - tecniche di progettazione di algoritmi
    - algoritmi "difficili"

La complessità come “raffinamento” della risolvibilità

- Non ci accontentiamo più di sapere se sappiamo risolvere (algoritmicamente) un problema, ma vogliamo sapere quanto ci costa risolverlo
- Analisi critica del concetto di “costo” (e beneficio):
  - Costo di esecuzione (risorse fisiche necessarie), a sua volta diviso in:
    - Tempo
      - di compilazione
      - di esecuzione
    - Spazio
  - Costo di sviluppo
  - …
  - Valutazioni oggettive e soggettive, trade-off tra obiettivi contrastanti, …
  - … verso problematiche e approcci da Ingegneria del software
- Qui ci si limita a concetti di costo oggettivi e formalizzabili:
  - tipiche risorse: memoria e tempo (di esecuzione)

Sarebbe bello partire come per la risolvibilità:

- Conseguenza fondamentale della Tesi di Church: Le domande che ci poniamo e le risposte che otterremo non dipendono dal modo con cui formuliamo il problema né dallo strumento usato.
- Però:
  - Fare la somma in unario è ben diverso dal fare la somma in base k
  - Se uso la tecnica z $\in L_\tau = \{x\$y \mid y = \boldsymbol{\tau}(x)\}$ per calcolare τ(x) dovrò decidere un problema di appartenenza un numero anche illimitato di volte per risolvere il problema originario di traduzione
  - E' verosimile che cambiando calcolatore (o MT) non cambi il tempo di esecuzione? Evidentemente no, però...

- Certo l’obiettivo è arduo o addirittura mal posto
- Tuttavia alla fine riusciremo ad ottenere risultati di notevole validità generale ….
- … una sorta di “Tesi di Church della complessità”
- Visto che per ora una Tesi di Church della complessità non sussiste …

Cominciamo da un’analisi di complessità legata alle MT

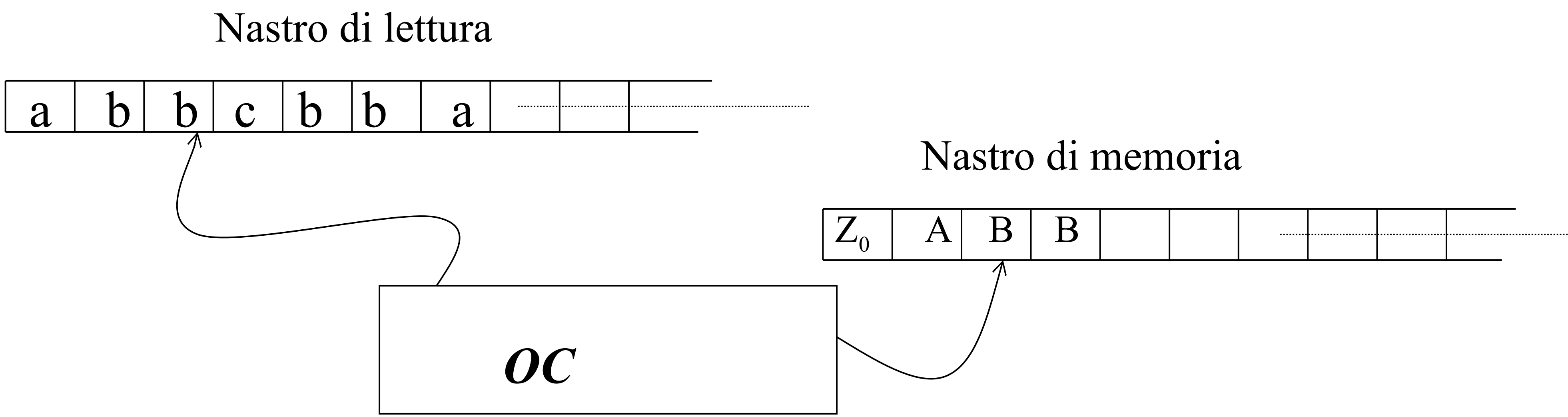
- *Complessità temporale*: sia
  $c_0 \vdash c_1 \vdash c_2 \vdash c_3 \ldots \vdash c_r$
  $T_M(x) = r$ se la computazione termina in c_r, altrimenti ∞

  importante: stiamo considerando MT **deterministiche**, quindi la computazione in corrispondenza di un input x è **unica**

  vedremo negli approfondimenti alla fine del corso che relazione c’è dal punto di vista della complessità tra modelli deterministici e nondeterministici

- *Complessità spaziale*: $c_0 \vdash c_1 \vdash c_2 \vdash c_3 \ldots \vdash c_r$
- $S_M(x) = \Sigma_{j\,=\,1..k} \max\{|\alpha_{ij}|+1 \mid i = 1..r\}$
  dove α_{ij} è il contenuto del nastro j alla mossa i-esima
  NB: $S_M(x) / k \leq T_M(x)$ per ogni x

Un primo esempio: riconoscimento di $\{wcw^R\}$



$T_M(x) = |x| + 1$ se $x \in L$

$|w|+1$ se $x = wz$, $w = vucu^R$, $v = \alpha a$, $z = b\alpha'$

$|x| + 1$ se $x \in \{a,b\}^*$ …

$S_M(x) = |x| + 1$ se $x \in \{a,b\}^*$, $\lfloor |x|/2 \rfloor + 1$ se $x \in L$, ...

- Un po' troppi dettagli, …
  - utili/necessari?
- Cerchiamo di semplificare e di andare al sodo:

- Complessità in $f(x) \rightarrow$ complessità in $f(n)$, con n "dimensione dei dati in ingresso":
  - n = $|x|$,
    righe/colonne di una matrice,
    numero di record in un file, …
- Però in generale

$$|x_1| = |x_2| \text{ non implica } T_M(x_1) = T_M(x_2)$$

(idem per S_M)

========>

- Scelta del caso pessimo:
  $T_M(n) = \max\{T_M(x), |x| = n\}$ (idem per $S_M(n)$)

- Scelta del caso ottimo:
  $T_M(n) = \min\{T_M(x), |x| = n\}$

- Scelta del caso medio:
  $T_M(n) = (\Sigma_{|x| = n} T_M(x)) / k^n$
  dove k è la cardinalità dell'alfabeto
  i.e. somma dei tempi per parole di lunghezza n / n. di parole di lunghezza n

- Noi adotteremo per lo più il *caso pessimo*:
- ingegneristicamente più rilevante (per certe applicazioni)
- matematicamente più semplice:
  - a rigore il caso medio dovrebbe tener conto di ipotesi probabilistiche sulla distribuzione dei dati: i nomi di una guida del telefono non sono equiprobabili

Tasso di crescita

- Il valore esatto di $T_M(n)$ non è di grande interesse per noi, ma solo il suo tasso di crescita rispetto a n
  - non è importante se $T_M(n)=3n^2+12n+35$ o se $T_M(n)=6n^2+26$, in quanto entrambi, per n abbastanza grande, si comportano in modo simile a n^2
  - non ci interessa neanche distinguere tra $T_M(n)=(1/50)n^2+1$ da $T_M(n)=1048n^2$!
  - Invece vogliamo separare $T_M(n)=3n^3+2$ da $T_M(n)=12n^2+22n$

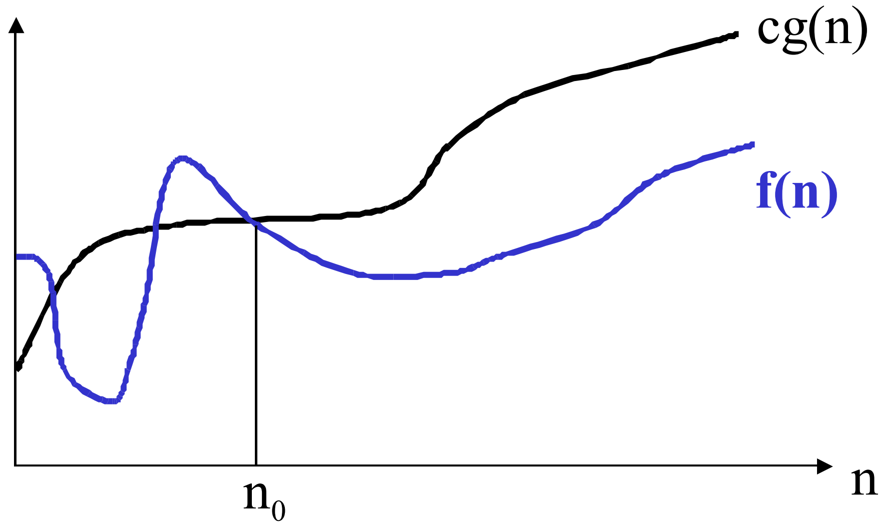
Tasso di crescita

- Ci interessa il comportamento asintotico delle funzioni di costo, non la loro espressione esatta

- Introduciamo opportune notazioni per mettere in evidenza il comportamento asintotico delle funzioni:
  - notazione O-grande (O)
  - notazione Omega-grande (Ω)
  - notazione Teta-grande (Θ)

Notazione O-grande

- La notazione O-grande indica un limite asintotico superiore
- Data una funzione g(n), O(g(n)) è il seguente insieme di funzioni:
  - $O(g(n)) = \{f(n) \mid \exists c,n_0 (c,n_0>0 \text{ tale che } \forall n>n_0 \;\; 0 \le f(n) \le cg(n))\}$
- Cioè:



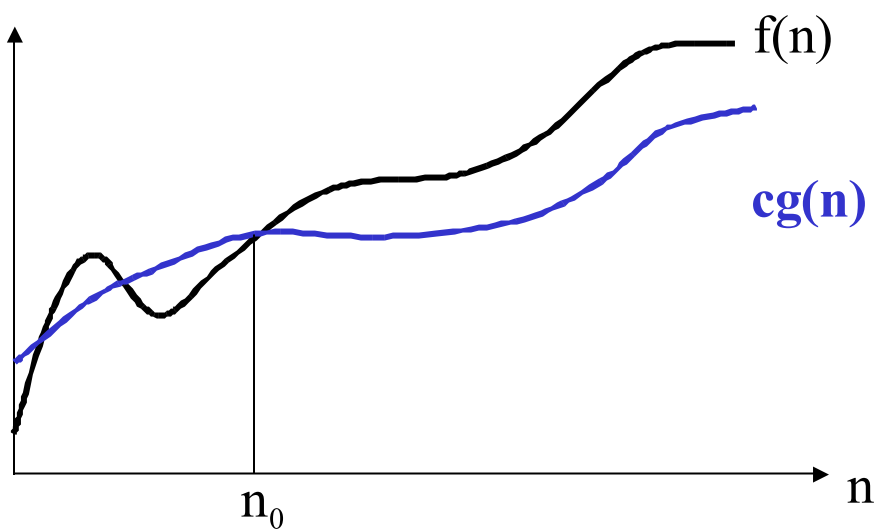
Esempi

- Abbiamo che:
  - $3n^2+12n+35 \in O(n^2)$
  - $5n^3+2 \in O(n^3)$
  - $2\log(n)+\log(\log(n)) \in O(\log(n))$
- Ma anche:
  - $3n^2+12n+35 \in O(n^3)$
  - $5n^3+2 \in O(n^{10})$
  - $5n^3+2 \in O(2^n)$
  - $2\log(n)+\log(\log(n)) \in O(n)$

- Spesso scriveremo "$f(n) = O(g(n))$" (oppure che "$f(n)$ è $O(g(n))$") invece di "$f(n) \in O(g(n))$"
  - $3n^2+12n+35 = O(n^2)$, $5n^3+2$ è $O(n^{10})$, ecc.

Notazione Omega-grande

- La notazione Omega-grande indica un limite asintotico inferiore
- Data una funzione g(n), $\Omega(g(n))$ è il seguente insieme di funzioni:
  - $\Omega(g(n)) = \{f(n) \mid \exists c, n_0 (c, n_0 > 0 \text{ tale che } \forall n > n_0 \;\; 0 \leq cg(n) \leq f(n))\}$
- Cioè:

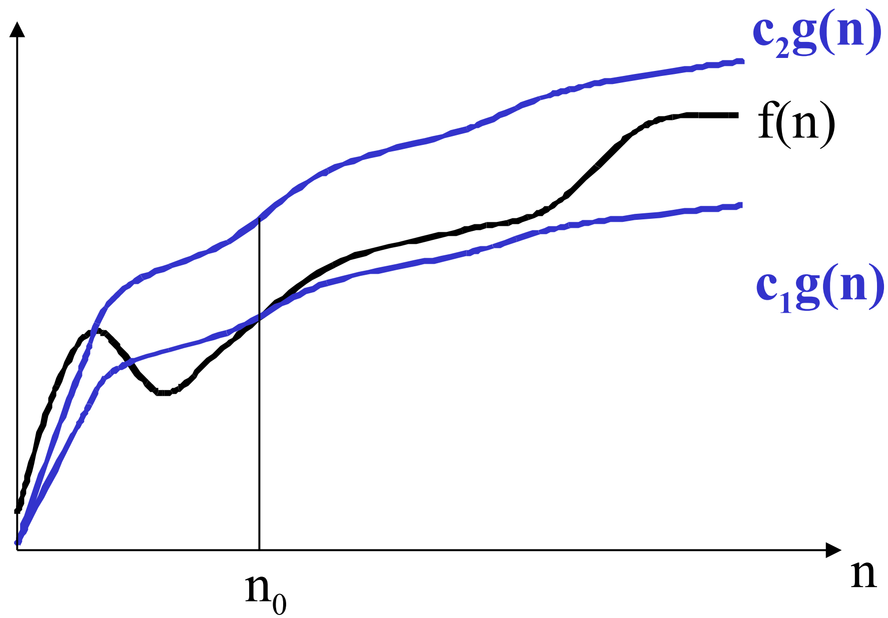


Esempi

- Abbiamo che:
  - $3n^2+12n+35=\Omega(n^2)$
  - $2n^2\log(n)+5=\Omega(n^2 \log(n))$
  - $n2^n+n^{45}=\Omega(n2^n)$
- Ma anche:
  - $3n^2+12n+35=\Omega(n)$
  - $3n^2+12n+35=\Omega(\log(n))$
  - $n2^n+n^{45}=\Omega(2^n)$
  - $n2^n+n^{45}=\Omega(n^{100})$
- Però
  - $3n^2+12n+35\neq\Omega(n^2\log(n))$
  - $2n^2\log(n)+5\neq\Omega(n^3)$

Notazione Teta-grande

- La notazione Teta-grande indica un limite asintotico che è sia inferiore che superiore
- Data una funzione g(n), $\Theta(g(n))$ è il seguente insieme di funzioni:
  - $\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2, n_0 (c_1, c_2, n_0 > 0$ tale che $\forall n > n_0\ \ 0 \le c_1 g(n) \le f(n) \le c_2 g(n))\}$
- Cioè:



Esempi

- Abbiamo che:
  - $3n^2+12n+35=\Theta(n^2)$
  - $2n^2\log(n)+5=\Theta(n^2\log(n))$
  - $n2^n+n^{45}=\Theta(n2^n)$
- Però
  - $3n^2+12n+35\neq\Theta(n)$
  - $3n^2+12n+35\neq\Theta(\log(n))$
  - $n2^n+n^{45}\neq\Theta(2^n)$
  - $n2^n+n^{45}\neq\Theta(n^{100})$

Alcune proprietà delle relazioni O, Ω, Θ

- $f(n) = \Theta(g(n))$ se e solo se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$
- Transitività
  - se $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n))$ allora $f(n) = \Theta(h(n))$
  - se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ allora $f(n) = O(h(n))$
  - se $f(n) = \Omega(g(n))$ e $g(n) = \Omega(h(n))$ allora $f(n) = \Omega(h(n))$
- Riflessività
  - $f(n) = \Theta(f(n))$
  - $f(n) = O(f(n))$
  - $f(n) = \Omega(f(n))$
- Simmetria: $f(n) = \Theta(g(n))$ se e solo se $g(n) = \Theta(f(n))$
- Simmetria trasposta: $f(n) = O(g(n))$ se e solo se $g(n) = \Omega(f(n))$

- Si noti che Θ è una *relazione di equivalenza*

- Si noti che se $\lim_{n\to\infty}\frac{f(n)}{g(n)} = c, c \neq 0, c \neq \infty$

  allora f(n) = Θ(g(n))
  - e quindi anche f(n) = O(g(n))

- Se $\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$

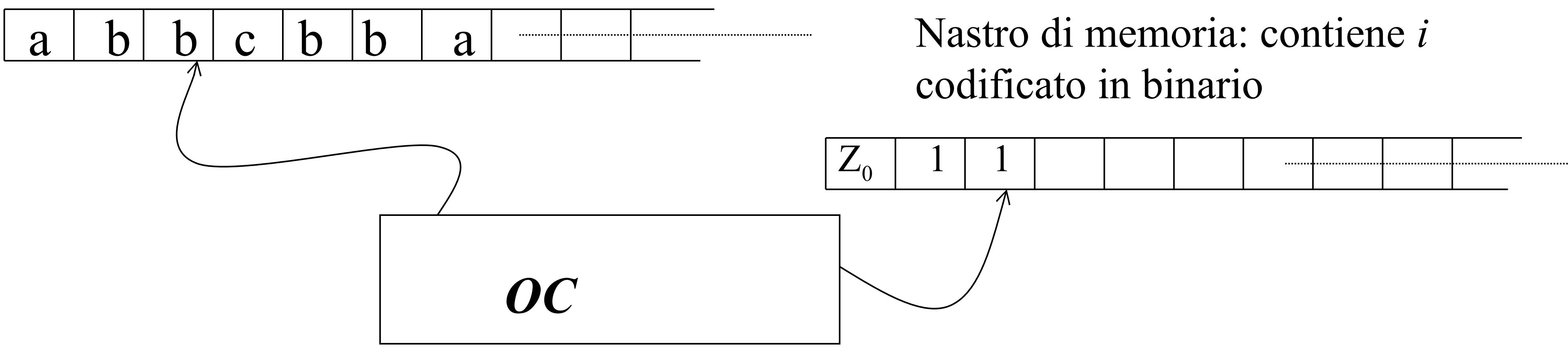
  allora f(n) = O(g(n)), ma f(n) ≠ Θ(g(n))
  - cioè g(n) ≠ O(f(n)); diremo in questo caso che Θ(f(n)) < Θ(g(n))

L'uso dell'ordine di grandezza permette di evidenziare con facilità la parte più importante di una funzione di complessità

Torniamo all'esempio $\{wcw^R\}$

- $T_M(n)$ è $\Theta(n)$, $S_M(n)$ è pure $\Theta(n)$
- Si può fare di meglio?
- Per $T_M(n)$ difficile (in generale dovrò almeno leggere tutta la stringa)
- Per $S_M(n)$:

Nastro di lettura



Memorizzo solo la posizione i del carattere da esaminare; poi sposto la testina di lettura in posizione i e n-i+1 e confronto i due caratteri letti ===>

Pseudo-codice per wcw^R con complessità spaziale logaritmica
[complessità del passo fra quadre]

- ciclo alla ricerca di "c": ad ogni passo si incrementa il numero in base 2 del nastro A [n log n]

Loop : [i va da n/2 a 0]

  - copia A nel nastro B [log i]
  - decrementa B passo passo fino a 0, ad ogni passo sposta verso sx la testina di ingresso [i log i]
  - leggi il carattere e memorizzalo in un nastro C [cost]
  - ritorna al carattere "c" [i]
  - copia A in B [log i]
  - decrementa B passo passo fino a 0, ad ogni passo sposta verso dx la testina di ingresso [i log i]
  - leggi il carattere e confrontalo con quello in C: se diversi HALT [cost]
  - decrementa A [log i]
  - ritorna al carattere "c" [i]
  - salta a Loop

- $S_M(n)$: $\Theta(\log(n))$

  ma
- $T_M(n)$: $\Theta(n^2 \log(n))$

- classico trade-off spazio-temporale
- L'esempio ci spiega anche perché nella MT a k nastri la testina di ingresso può muoversi nelle due direzioni: in caso contrario non ci sarebbero esempi significativi di complessità spaziale sublineare

A proposito di MT a k nastri:

- Proviamo a cambiare modello di calcolo (sempre <u>deterministico</u>):
- FSA hanno sempre $S_A(n)$ $\Theta(k)$ (costante) e $T_A(n)$ $\Theta(n)$
  - anzi $T_A(n) = n$ (macchine real-time);
- PDA hanno sempre $S_A(n) \leq \Theta(n)$ e $T_A(n)$ $\Theta(n)$

- MT a nastro singolo?
- Il riconoscimento di $\{wcw^R\}$ richiede in prima istanza $\Theta(n^2)$
- La complessità spaziale non potrà mai essere < $\Theta(n)$
  (ciò fornisce un'ulteriore spiegazione della scelta della MT a k nastri come modello principale)
- Si può fare meglio di $\Theta(n^2)$? NO!
  - dimostrazione tecnicamente complessa come quasi sempre per limiti inferiori di complessità che non siano banali.
- MT a nastro singolo più potenti dei PDA ma talvolta meno efficienti
- e i calcolatori *a la* von Neumann? Lo vedremo più avanti.

I teoremi di “accelerazione” lineare

- Se L è accettato da una MT M a k nastri con complessità $S_M(n)$, per ogni c > 0 ($c \in \mathbf{R}$) si può costruire una MT M’ (a k nastri) con complessità $S_{M'}(n) < c\, S_M(n)$

			a_1	a_2		a_i		a_r	b_1	b_2		b_i		b_r	c_1	c_2		c_i		

Ovviamente c’è il problema della memorizzazione della posizione della testina: si può arricchire l’alfabeto oppure memorizzare la posizione relativanello stato dell’organo di controllo

			$<a_1 \ldots\ a_i \ldots a_r>$	$<b_1 \ldots\ b_i \ldots b_r>$	$<c_1 \ldots\ c_i\ \ldots c_r>$	

Prendiamo un r tale che sia r c > 2

- Se L è accettato da una MT M a k nastri con complessità $S_M(n)$, si può costruire una MT M' a 1 nastro (*non* a nastro singolo) con complessità
  $S_{M'}(n) = S_M(n)$.

(si mettono le parti significative dei k nastri nell'unico nastro, una dietro l'altra)

- Se L è accettato da una MT M a k nastri con complessità $S_M(n)$, per ogni $c > 0$ ($c \in \mathbf{R}$) si può costruire una MT M' a 1 nastro con complessità
  $S_{M'}(n) < c\, S_M(n)$.

(idem + codifica)

- Se L è accettato da una MT M a k nastri con complessità $T_M(n)$, per ogni $c > 0$ ($c \in \mathbf{R}$) si può costruire una MT M' (a k+1 nastri) con complessità
  $T_{M'}(n) = \max\{n+1, c\, T_M(n)\}$
- Schema di dimostrazione analogo a quello usato per la complessità spaziale. Però, con qualche dettaglio tecnico in più:
  - occorre prima leggere e tradurre tutto l'input (richiede n mosse)
  - ciò crea qualche problema all'interno della classe $\Theta(n)$ (di quì il *max{n+1 ...}*)
  - nel caso pessimo occorrono 3 mosse per simulare almeno r + 1 mosse di M

es: $<a_1..a_r><b_1..b_r><c_1..c_r>$

- caso pessimo: da a_r arrivo a b_r con r mosse; con una mossa in più sono riuscito a toccare 3 celle (= 3 mosse della M')

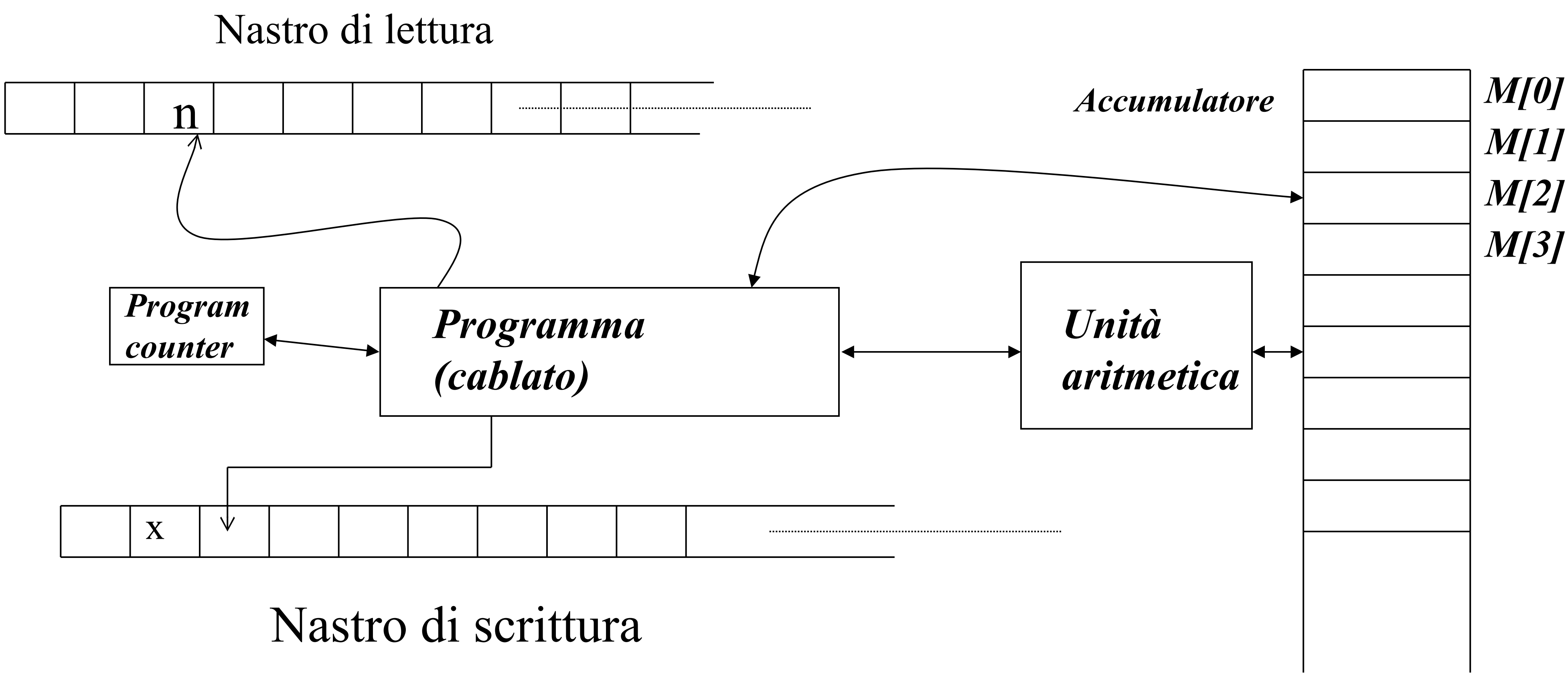
Conseguenze pratiche dei teoremi di accelerazione lineare

- Lo schema di dimostrazione è valido per qualsiasi tipo di modello di calcolo, quindi anche per calcolatori reali:
  - significa aumentare il parallelismo fisico (da 16 bit a 32 a 64 …)
- pur di aumentare la potenza di calcolo in termini di risorse disponibili si può aumentare "a piacere" la velocità di esecuzione
- però tale aumento di prestazioni rimane confinato nell'ambito di miglioramenti al più lineari
  - non riesco a cambiare l'ordine di grandezza
- miglioramenti di ordini di grandezza possono essere ottenuti solo cambiando algoritmo e in modo non automatico:
- l'intelligenza può superare di gran lunga la forza bruta!

Riprendiamo ora il confronto tra MT e calcolatori reali

- A prima vista il confronto è impari …
  - per calcolare la somma di due numeri una MT impiega $\Theta(n)$ (n è la lunghezza -della stringa di caratteri che codifica- i due numeri) mentre un calcolatore fornisce questa operazione come elementare (eseguita in un ciclo macchina)
  - un calcolatore può accedere direttamente a una cella di memoria, mentre la MT ha accesso solo sequenziale.
- Non possiamo perciò accontentarci di valutazioni di complessità legate esclusivamente alle MT

Un modello molto astratto di calcolatore: la RAM



Ogni cella contiene un intero, non un carattere!

- Il repertorio istruzioni della RAM:

– LOAD	X	M[0] := M[X]	ind. *diretto*
– LOAD=	X	M[0] := X	ind. *immediato*
– LOAD*	X	M[0] := M[M[X]]	ind. *indiretto*
– STORE [*]	X	M[X] := M[0], …	
– ADD …		M[0] := M[0]+M[X], …	
– SUB, MULT, DIV			
– READ [*]	X		
– WRITE [=, *]	X		
– JUMP	lab	PC := b(lab)	
– JZ, JGZ, ...	lab	jump if M[0] = 0, > 0, ...	
– HALT			

Un programma RAM che calcola la funzione *is_prime(n) =* ***if*** *n is prime* ***then*** *1* ***else*** *0*

```
        READ    1      Il valore di ingresso n è memorizzato nella cella M[1]
        LOAD=   1      Se n = 1, esso banalmente non è primo ...
        SUB     1
        JZ      NO
        LOAD=   2      M[2] è inizializzato a 2
        STORE   2
LOOP:   LOAD    1      Se M[1] = M[2] allora n è primo
        SUB     2
        JZ      YES
        LOAD    1      Se M[1] = (M[1] div M[2]) * M[2] allora
        DIV     2      M[2] è un divisore di M[1];
        MULT    2      quindi M[1] non è primo
        SUB     1
        JZ      NO
        LOAD    2      M[2] è incrementato di 1 e il ciclo viene ripetuto
        ADD=    1
        STORE   2
        JUMP    LOOP
YES:    WRITE=  1
        HALT
NO:     WRITE=  0
        HALT
```

Quanto costa eseguire il programma di cui sopra mediante una RAM?

- Assunzione di base (per ora): ogni istruzione ha un *costo costante*, indipendentemente dal valore degli operandi
  - indichiamo con c_i il costo della istruzione i-esima del programma
- Costo del programma RAM:
  - Le istruzioni da 1 a 6 vengono eseguite al più una volta; il loro costo è $c_1+c_2+c_3+c_4+c_5+c_6$ ed è una costante (chiamiamola k_1)
  - Le istruzioni da 19 a 22 anch'essere vengono eseguite al più una volta sola, ed il loro costo è
    $c_{19}+c_{20}+c_{21}+c_{22}$, anch'esso costante (chiamiamo k_3 la costante)
  - Le istruzioni dalla 7 alla 18 hanno costo
    $c_7+c_8+c_9+c_{10}+c_{11}+c_{12}+c_{13}+c_{14}+c_{15}+c_{16}+c_{17}+c_{18}=k_2$
    esse però vengono eseguite, nel caso pessimo, n volte
- Quindi

  $T_R(n) = k_1+k_2n+k_3 = \Theta(n)$

  $S_R(n) = 2 = \Theta(1)$, perché vengono occupate solo 2 celle di memoria, M[1], ed M[2]
- Attenzione però: che cos'è n?
  - Non è la lunghezza della stringa di ingresso!
  - Attenzione al parametro di "dimensione dei dati"!

- Inoltre:
  - Riconoscimento di wcw^R con
    - $S_R(n) = \Theta(n)$
    - $T_R(n) = \Theta(n)$

- Algoritmo di *ricerca binaria*
  - precondizione: l'input è una sequenza *ordinata* di numeri (interi), ed un numero da cercare
  - postcondizione: l'algoritmo restituisce 1 se l'elemento cercato esiste nella sequenza, 0 altrimenti

Ricerca binaria

- Assunzione: numeri già in memoria quando viene eseguito questo pezzo di codice
  - M[1]: cella contenente l'indirizzo del primo numero della sequenza
  - M[2]: cella contenente il numero di elementi della sequenza
    - quindi la sequenza va dalla cella M[M[1]] alla cella M[M[1]+M[2]-1]

	READ	3	L'elemento da cercare è letto e memorizzato in M[3]
	LOAD	1	
	STORE	4	M[4] è inizializzato con l'indirizzo del primo numero
	ADD	2	
	SUB=	1	
	STORE	5	M[5] è inizializzato con l'indirizzo dell'ultimo numero
LOOP:	LOAD	5	Se l'indirizzo di M[5] precede quello di M[4] siamo
	SUB	4	giunti ad avere una porzione di sequenza che è vuota,
	JLZ	NO	quindi l'elemento cercato non esiste
	LOAD	5	
	ADD	4	
	DIV=	2	
	STORE	6	M[6] ora contiene l'indirizzo dell'elemento centrale

```
        LOAD*   6
        SUB     3
        JZ      YES     Se M[3]=M[M[6]], l’elemento cercato esiste
        JGZ     FST     Se M[3]<M[M[6]], cerca nella prima metà
        JLZ     SND     Se M[3]>M[M[6]], cerca nella seconda metà
FST:    LOAD    6
        SUB=    1
        STORE   5       Memorizza M[6]-1 in M[5]
        JUMP    LOOP
SND:    LOAD    6
        ADD=    1
        STORE   4       Memorizza M[6]+1 in M[4]
        JUMP    LOOP
YES:    WRITE=  1
        HALT
NO:     WRITE=  0
        HALT
```

- Alla peggio il ciclo viene eseguito $\log_2(n)$ volte, in quanto ogni volta la dimensione dell’array viene dimezzata
- Quindi $T_R(n) = \Theta(\log(n))$ (mentre per la MT non si va sotto $\Theta(n)$)

Un problema: calcoliamo 2^{2^n} usando una RAM (o macchina analoga)

```
read n;
x = 2;
for (i = 1; i <= n; i++)
    x = x*x;
write x
```

- Ottengo $((2)^2)^2)$ … n volte ossia 2^{2^n}
- Quale complessità temporale?
  - $\Theta(n)$!!
- Siamo proprio sicuri?
  - In realtà occorrono almeno 2^n bit solo per scrivere il risultato
- L'analisi sembra decisamente irrealistica!

Il problema sta nel fatto che la RAM è un po' troppo astratta

- Una cella contenente un numero arbitrario = unità di memoria?
- Un'operazione aritmetica = operazione elementare a costo unitario?
- Ciò è corretto solo fino a quando la macchina reale (a 16, 32, 64, … bit) corrisponde esttamente alla macchina astratta.
- Altrimenti … doppia precisione ecc. ---> le operazioni relative non sono più elementari e occorre programmarle ad hoc.

- Quindi rifacciamo tutti gli algoritmi e le relative analisi di complessità in funzione del livello di precisione (numero di bit) usati?
- Concettualmente sì ma più comodamente:
- **Criterio di costo logaritmico**, basato su un'analisi "microscopica" (vedi "microcodice") delle operazioni HW

- Quanto costa copiare il numero i da una cella all’altra?
  - tante microoperazioni elementari quanti sono i bit necessari a codificare i: log(i)
- Quanto costa accedere alla cella di posizione i-esima?
  - l’apertura di log(i) “cancelli” di accesso ad altrettanti banchi di memoria
- Quanto costa eseguire l’operazione
  LOAD i?
- …
- Con semplice e sistematica analisi si ottiene la seguente ...

Tabella dei costi logaritmici della RAM

LOAD=	x	$l(x)$
LOAD	x	$l(x) + l(M[x])$
LOAD*	x	$l(x) + l(M[x]) + l(M[M[x]])$
STORE	x	$l(x) + l(M[0])$
STORE*	x	$l(x) + l(M[x]) + l(M[0])$
ADD=	x	$l(M[0]) + l(x)$
ADD	x	$l(M[0]) + l(x) + l(M[x])$
ADD *	x	$l(M[0]) + l(x) + l(M[x]) + l(M[M[x]])$
…		
READ	x	l(valore di input corrente) + $l(x)$
READ*	x	l(valore di input corrente) + $l(x) + l(M[x])$
WRITE=	x	$l(x)$
WRITE	x	$l(x) + l(M[x])$
WRITE*	x	$l(x) + l(M[x]) + l(M[M[x]])$
JUMP	lab	1
JGZ	lab	$l(M[0])$
JZ	lab	$l(M[0])$
HALT		1

$l(i) :=$ if $i=0$ then 1
else $\lfloor \log_2 |i| \rfloor + 1$

Applicando il nuovo criterio di costo

- Al calcolo di *is-prime(n)* (solo nei punti essenziali)

```
7  LOOP:   LOAD   1     1+l(n)
8          SUB    2     l(n) +2 + l(M[2])
9          JZ     YES   l(M[0])
10         LOAD   1     1+l(n)
11         DIV    2     l(n) +2 + l(M[2])
12         MULT   2     l(n/M[2]) +2 + l(M[2])      < l(n)
13         SUB    1     l(M[0]) +1 + l(n) < 2 l(n) +1
14         JZ     NO    ≤ l(n)
15         LOAD   2     ≤ l(n) + k
16         ADD=   1     ...
17         STORE  2
18         JUMP   LOOP
```

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

- Similmente otteniamo:
  - per il riconoscimento di wcw^R: $\Theta(n \log(n))$
    - NB: più della MT! E' possibile fare meglio?
  - per la ricerca binaria: $\Theta(\log^2(n))$ …

- Costo a criterio di costo logaritmico = Costo a criterio di costo costante · log(n)?
- Costo a criterio di costo logaritmico =
  Costo a criterio di costo costante · log(Costo a criterio di costo costante)?

- Spesso ma non sempre: per il calcolo di 2^{2^n} costo a criterio di costo logaritmico 2^n, infatti complessità temporale >= complessità spaziale, che qui è $\Theta(2^n)$

- Esiste un criterio per scegliere il criterio?
- A 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
    => non dipende dai dati
    => una singola cella è considerabile elementare e con essa le operazioni relative
    => criterio di costo costante OK
  - Altrimenti (fattoriale, 2^{2^n}, ricorsioni "feroci", …) indispensabile criterio logaritmico: l'unico "garantito"!

Le relazioni tra le complessità relative ai diversi modelli di calcolo

- Lo stesso problema risolto con macchine diverse può avere complessità diverse
  - Può darsi che per P1 il modello M1 sia meglio del modello M2 ma per P2 succeda il contrario
    - ricerca binaria → accesso diretto
    - riconoscimento di wcw^R → accesso e memorizzazione sequenziale
- Non esiste un modello migliore in assoluto
- Non esiste un analogo della tesi di Church per la complessità
- Però è 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 (il criterio di costo costante per la RAM non è “ragionevole” in assoluto!):
- *Se un problema è risolvibile mediante un modello di calcolo M_1 con complessità (spazio/temporale) $C_1(n)$, allora è risolvibile da qualsiasi altro modello di calcolo M_2 con complessità $C_2(n) \leq P_2(C_1(n))$, essendo P_2 un opportuno polinomio*

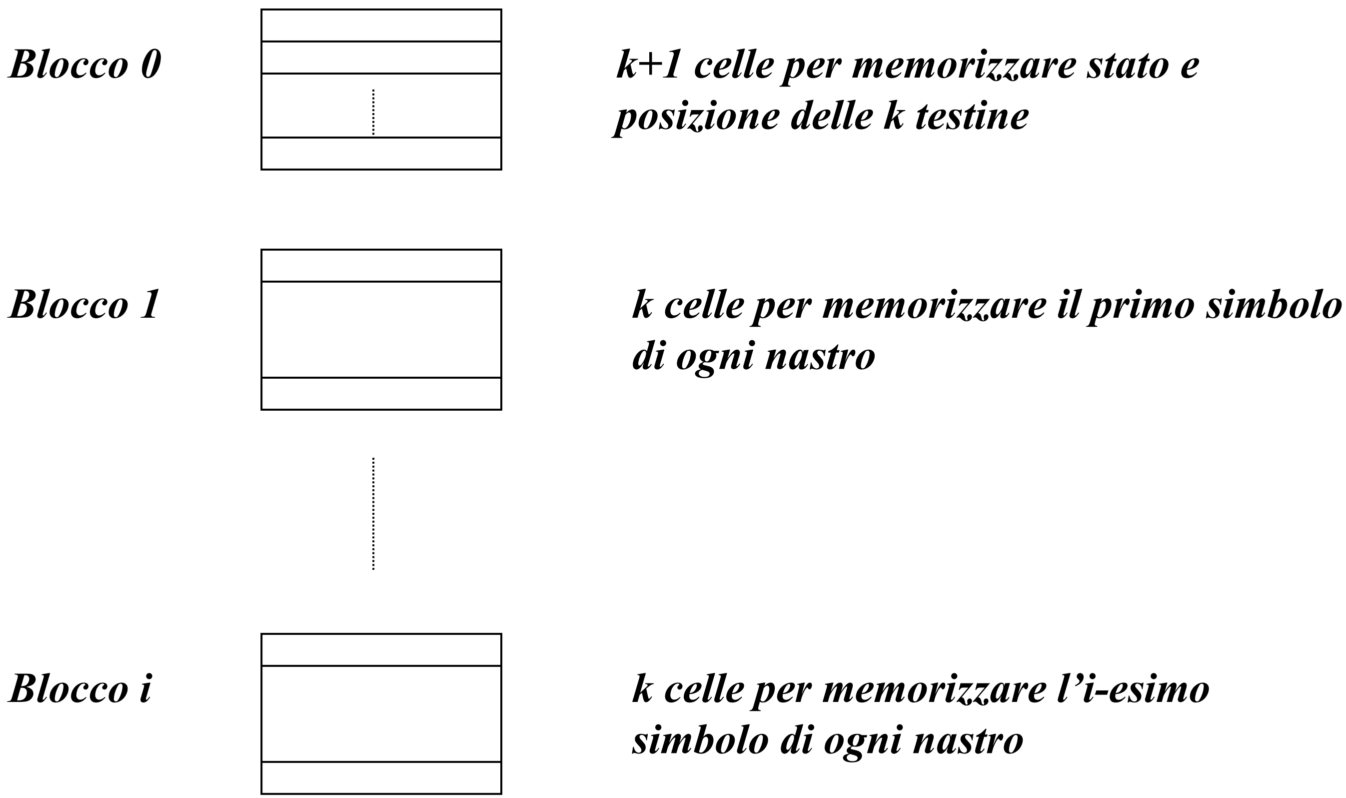
- è vero che i polinomi possono anche essere n^{1000}, ma è sempre meglio dell’"abisso" esponenziale (n^k contro 2^n)
- Grazie al teorema di correlazione polinomiale possiamo parlare della *classe dei problemi risolvibili in tempo/spazio polinomiale* (non di quelli risolvibili in tempo quadratico!): la classe non dipende dal modello adottato

Prima di dimostrare il teorema (non più *tesi*!) nel caso MT-RAM, valutiamone l'impatto:

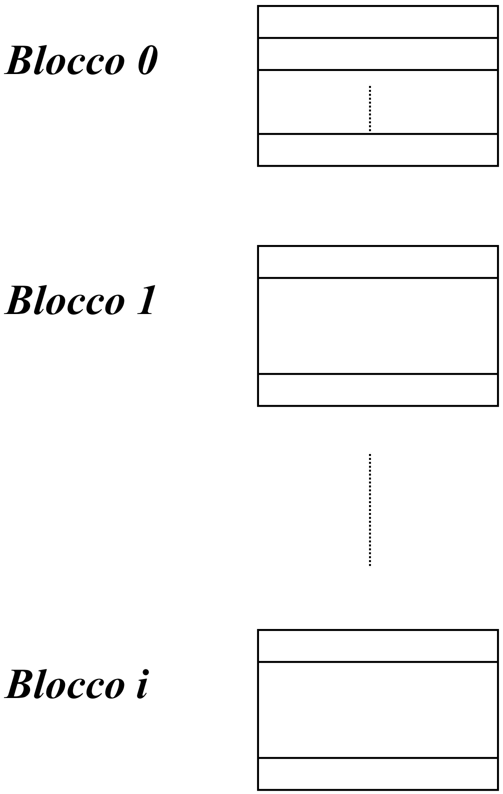
- Sostanzialmente grazie a questo risultato si è da tempo adottata l'analogia:
  - classe dei problemi "trattabili" in pratica = classe dei problemi risolvibili in tempo polinomiale con modelli deterministici: **P**
  - La teoria include in **P** anche i problemi con complessità n^{1000} (comunque sempre meglio di quelli a complessità esponenziale), ma l'esperienza pratica conferma che i problemi di interesse applicativo (ricerche, cammini, ottimizzazioni, …) che sono in **P** hanno anche grado del polinomio accettabile
  - (similmente vedremo tra poco che la relazione di complessità tra MT e RAM è "piccola")

La correlazione (temporale) tra MT e RAM:
1: Da MT (a k nastri) a RAM

- La memoria della RAM simula la memoria della MT:
  1 cella RAM per ogni cella di nastro di MT
  Però, invece di usare blocchi di memoria RAM per simulare ogni nastro, associamo un blocco -di k celle- ad ogni k-pla di celle prese per ogni posizione di nastro, + un blocco "di base":



Una mossa della MT simulata dalla RAM:



Lettura:

- Viene esaminato il contenuto del blocco 0 (pacchetto di k+1 accessi, c(k+1) mosse)
- Vengono fatti k accessi indiretti in k blocchi per esaminare il contenuto delle celle in corrispondenza delle testine

Scrittura:

- Viene cambiato lo stato
- Vengono aggiornati, mediante STORE indiretti, i contenuti delle celle corrispondenti alla posizione delle testine
- Vengono aggiornati, nel blocco 0, i valori delle posizioni delle k testine

Una mossa di MT richiede h · k mosse di RAM:

- A criterio di costo costante T_R è $\Theta(T_M)$
- A criterio di costo logaritmico (quello “serio”)
  T_R è $\Theta(T_M log(T_M))$
  (un accesso indiretto a i costa log(i))

La correlazione (temporale) tra MT e RAM:
2: Da RAM a MT

(in un caso semplice ma centrale: riconoscimento di linguaggi senza usare MULT e DIV - la generalizzazione è banale)

- Il nastro principale della MT:

....	**\$**	$\boldsymbol{i_j}$	**#**	$\boldsymbol{M[i_j]}$	**\$**	....	**\$**	$\boldsymbol{i_k}$	**#**	$\boldsymbol{M[i_k]}$	**\$**	....

- NB:
  - Le varie celle RAM sono tenute in ordine
  - Inizialmente il nastro è vuoto ---> in un generico istante vi si trovano memorizzate solo le celle che hanno ricevuto un valore (tramite una STORE)
  - i_j e $M[i_j]$ sono rappresentati in codifica binaria
- Ulteriori nastri:
  - Un nastro contiene M[0] (in binario)
  - Un nastro di servizio

- Una mossa della RAM è simulata dalla MT:

…………	**$**	$\boldsymbol{i_j}$	**#**	$\boldsymbol{M[i_j]}$	**$**	…………	**$**	$\boldsymbol{i_k}$	**#**	$\boldsymbol{M[i_k]}$	**$**	…………

- Esaminiamone un campione:
- LOAD h
  - Si cerca il valore h nel nastro principale (se non si trova: errore)
  - Si copia la parte accanto, M[h] in M[0]
- STORE h
  - Si cerca h
  - Se non si trova si "crea un buco" usando il nastro di servizio. Si memorizza h e si copia M[0] nella parte accanto (M[h]); si ricopia la parte successiva dal nastro di servizio
  - Se h esiste già si copia M[0] nella parte accanto (M[h]); ciò può richiedere l'uso del nastro di servizio se il numero di celle già occupate non è uguale a quelle di M[0]
- ADD* h
  - Si cerca h; si cerca M[h]; …

- Con facile generalizzazione:
  - Simulare una mossa di RAM può richiedere alla MT un numero di mosse maggiorabile da *c · lunghezza del nastro principale*.

- Lemma: la lunghezza del nastro principale è limitata superiormente da una funzione $\Theta(T_R)$:
  - Ogni "cella i_j-esima" della RAM richiede nel nastro $l(i_j) + l(M[i_j])$ (+2) celle del nastro
  - Ogni "cella i_j-esima" esiste nel nastro se e solo se la RAM ha eseguito almeno una STORE su di essa.
  - La STORE è costata alla RAM $l(i_j) + l(M[i_j])$, quindi:
  - Per riempire r celle, di lunghezza complessiva $\sum_{j=1..r} l(i_j)+l(M[i_j])$, alla RAM occorre un tempo ($\leq T_R$) almeno proporzionale allo stesso valore.

- Dunque, per simulare una mossa della RAM, la MT impiega un tempo al più $\Theta(T_R)$
- una mossa di RAM costa *almeno* 1; se la RAM ha complessità T_R esegue al più T_R mosse (a costo costante sono T_R, a costo logaritmico sono *di meno*)
- quindi la simulazione completa della RAM da parte della MT costa *al più* $\Theta(T^2{}_R)$.

Alcune puntualizzazioni 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 MT viola il teorema di correlazione polinomiale?
  - Attenzione all'ipotesi: riconoscimento di linguaggio => dati non già in memoria => complessità almeno lineare.
- Operazioni dominanti (e.g. I/O): complessità lineare rispetto alle operazioni dominanti e quadratica in complesso?