La complessità del calcolo

- In questo corso:
  - Non analisi di singoli algoritmi (si fa riferimento ai corsi di fondamenti)
  - Non algoritmica avanzata (si rimanda a corsi successivi)
- Bensì:
  - Riesame critico del problema e dell'approccio
  - Ricerca di principi di validità generale
  - Costruire una capacità di inquadramento nel giusto ambito di singoli problemi

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:

- Le domande che ci poniamo e le risposte che otterremo non dipendono dal modo con cui formuliamo il problema né dallo strumento usato (Tesi di Church).
- Però:
  - Fare la somma in unario è ben diverso dal fare la somma in base k
  - Se uso la tecnica $z \in L_\tau = \{x\$y \mid y = \tau(x)\}$ per calcolare $\tau(x)$ 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 …

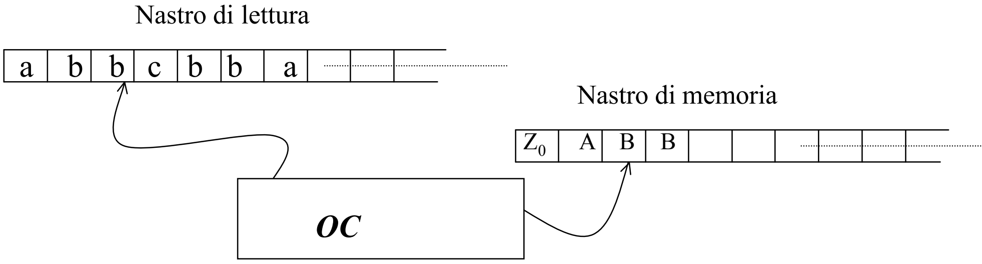
… cominciamo da un'analisi di complessità legata alle MT

- Complessità temporale: sia
  c_0 |-- c_1 |-- c_2 |-- c_3 … |-- c_r
  $T_M(x) = r$ se la computazione termina in c_r, altrimenti ∞
- Complessità spaziale:
  c_0 |-- c_1 |-- c_2 |-- c_3 … |-- c_r

  $S_M(x) = \sum_{j=1}^{k} \max\{|\alpha_{ij}| + 1 \mid i = 1,\dots,r\}$ con α_{ij} contenuto del nastro j alla mossa i-esima

- NB: $\frac{S_M(x)}{k} \le T_M(x) \quad \forall x$

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 $x \in L$, ...

7

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

- Complessità in f(x) → 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

  $$|x_1| = |x_2| \not\Rightarrow T_M(|x_1|) = T_M(|x_2|)$$

  (idem per S_M) --->

8

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

  $$T_M(n) = \frac{\sum_{|x|=n} T_M(x)}{k^n}, k = \text{cardinalità dell'alfabeto}$$

- 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

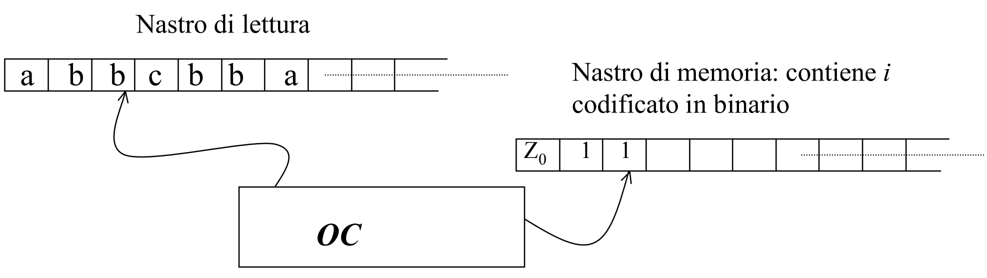
- Uso della notazione Θ per valutare l'ordine di grandezza di una funzione (di complessità)

$$f\Theta g \leftrightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)} = c, c \neq 0, c \neq \infty$$

- è una relazione di equivalenza ---->
- Diremo che $T_M(n)$ è $\Theta(n)$, ….
  - (Dire che $T_M(n)$ è $\Theta(n)$ è come dire che $T_M(n)$ è lineare?)
- L'uso dell'ordine di grandezza permette di
  - evidenziare con facilità la parte più importante di una funzione di complessità
  - in un certo senso esso descrive anche la parte "indipendente dalla potenza di calcolo"

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 $S_M(n)$:



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 ===>

- $S_M(n)$: $\Theta(\log(n))$
  ma
- $T_M(n)$: $\Theta(n^2\log(n))$:
  - $\forall$i,
    - costruisco i in binario (in un nastro) ($\Theta(\log(i))$ per implementare i := i+1);
    - copio i su un nastro ausiliario (j := i);
    - i volte:
      - decremento j di 1 e sposto di 1 la testina nel nastro di ingresso, a partire dalla c centrale (complessità temporale: $\Theta(\log(i))$);
      - quando j = 0 la testina è in posizione i-esima
    - complessità temporale totale del passo: $\Theta(i \log(i))$;
- 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

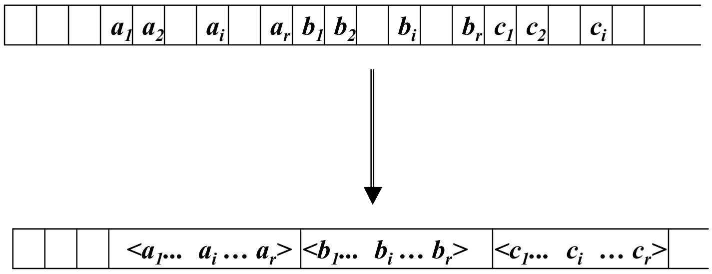
A proposito di MT a k nastri ...

- Proviamo a cambiare modello di calcolo:
  - 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?
  - Aspettiamo ancora un po' ...

I teoremi di “accelerazione” lineare

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



$r*c \geq 2$

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

- Se L è accettato da una MT M a k nastri con complessità $T_M(n)$, per ogni c > 0 ($c \in \Upsilon$) 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 $\Theta(n)$
  - nel caso pessimo occorrono 3 mosse per simulare almeno r + 1 mosse di 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:
  - per valori di n sufficientemente grandi ordinare una sequenza di n elementi con il merge sort sarà sempre più efficiente che ordinarla mediante inserzione lineare o bubble-sort, anche se il primo algoritmo viene eseguito su una macchina di modesta "potenza" e il secondo da un supercalcolatore
- l'intelligenza può superare di gran lunga la forza bruta!

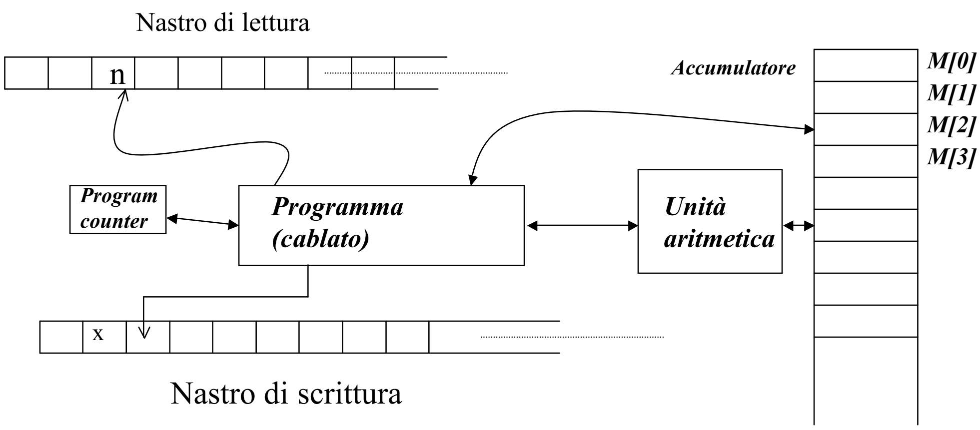
17

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 Θ(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:
    - ad esempio, se cerchiamo di implementare la ricerca binaria mediante una MT otteniamo addirittura una complessità Θ(n log(n)) > Θ(n)
- Non possiamo perciò accontentarci di valutazioni di complessità legate esclusivamente alle MT

18

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], X, M[M[X]]
  - 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
  - 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 è banalmente primo ...
	SUB	1	
	JZ	YES	
	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		

21

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

- Ovviamente:
  - $S_R(n)$ $\Theta(2)$
  - $T_R(n)$ $\Theta(n)$
  - Attenzione però: che cos'è n??
    - Non è la lunghezza della stringa di ingresso!
    - Attenzione al parametro di "dimensione dei dati"!!
- Pure ovviamente:
  - Riconoscimento di wcw^R con
    - $S_R(n)$ $\Theta(n)$
    - $T_R(n)$ $\Theta(n)$
  - Ricerca binaria con $T_R(n)$ $\Theta(\log(n))$
  - Ordinamento
  - …
- Però ...

22

Calcoliamo 2^{2^n} usando una RAM (o macchina analoga)

```
read n;
x := 2;
for i :=1 to n do 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 (macchina di von Neumann) 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.

$\longrightarrow$

- 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

25

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

Applicando il nuovo criterio di costo

26

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

LOOP:	LOAD	1	1+*l*(n)	
	SUB	2	*l*(n) +2 + *l*(M[2])	
	JZ	YES	*l*(M[0])	
	LOAD	1	1+*l*(n)	
	DIV	2	*l*(n) +2 + *l*(M[2])	
	MULT	2	*l*(n/M[2]) +2 + *l*(M[2])	< *l*(n)
	SUB	1	*l*(M[0]) +1 + *l*(n)	< 2 *l*(n) +1
	JZ	NO	≤ *l*(n)	
	LOAD	2	≤ *l*(n) + k	
	ADD=	1	...	
	STORE	2		
	JUMP	LOOP		

- In conclusione si può facilmente maggiorare la singola iterazione del ciclo con Θ (log(n))
- Ergo la complessità temporale complessiva è Θ(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 $\geq 2^n$
    - 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:
  - E' possibile stabilire almeno 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

29

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

- E' 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
- Grazie a questo risultato -e ad altri importanti fatti teorici- si è 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} (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")

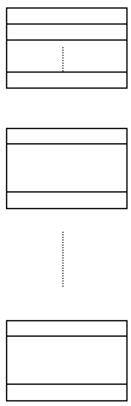
30

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":

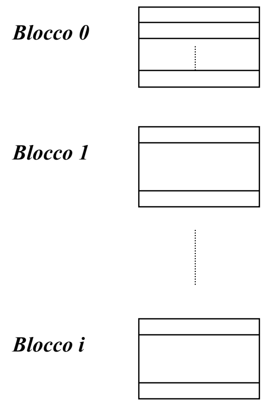


Blocco 0 — ***K+1 celle per memorizzare stato e posizione delle k testine***

Blocco 1 — ***K celle per memorizzare il primo simbolo di ogni nastro***

Blocco i — ***K celle per memorizzare l'i-esimo simbolo di ogni nastro***

Una mossa della MT è 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 corrispondnza 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:

.....	**$**	i_j	**#**	$M[i_j]$	**$**	.....	**$**	i_k	**#**	$M[i_k]$	**$**	.....

- 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:

………	**$**	i_j	#	$M[i_j]$	**$**	………	**$**	i_k	#	$M[i_k]$	**$**	………

- 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.*

- A questo punto:
- Lemma: la lunghezza del nastro principale è limitata superiormente da una funzione $\Theta(T_R)$

………	**$**	i_j	#	$M[i_j]$	**$**	………	**$**	i_k	#	$M[i_k]$	**$**	………

  - 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 $l(i_j) + l(M[i_j])$
    ---->
  - 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

$$\longrightarrow$$

- la simulazione completa della RAM da parte della MT costa *al più* $\Theta(T^2_R)$.

Alcune puntualizzazioni e avvertimenti conclusivi

- 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?
- Caso pessimo e caso medio nei *casi* pratici (Quicksort, compilazione, …: eccezioni?)

Apriamo infine -fuori programma- una piccola finestra su aspetti avanzati ma estremamente rilevanti della complessità del calcolo

- Alcune domande importanti:
  - Esistono limiti inferiori alla complessità?
  - Aumentando la complessità si aumenta (sempre) la classe dei problemi risolvibili? (se spendo di più ottengo di più?)
  - Esiste una sorta di "classe universale di complessità" (tutti i problemi risolvibili si possono risolvere all'interno di una certa classe)?
  - Come si definisce una "classe di complessità"?
  - Ha senso, e se sì, come, definire la complessità di macchine nondeterministiche?
  - L'introduzione del nondeterminismo può cambiare la complessità di soluzione dei problemi?
  - ...

37

Concentriamoci sulla computazione nondeterministica

- In primis: come si definisce?
  - La computazione più veloce?
  - La più lenta?
  - E se alcune computazioni non terminano e altre sì?
  - Solo le computazioni che accettano?
- La computazione più veloce tra quelle che accettano … se ce ne sono!

- Che significato pratico hanno le computazioni nondeterministiche, visto che le macchine reali sono deterministiche?
- Per rispondere rifacciamoci al tema generale di computazione nondeterministica: modello per parallelismo, ricerca "cieca" tra diverse vie, …
- Il grande impatto pratico di questo tema nasce proprio dal fatto ….

38

- … che moltissimi problemi di grande interesse pratico hanno semplice, naturale ed "efficiente" soluzione in termini nondeterministici:
  - Il cammino hamiltoniano in un grafo
  - La soddisfacibilità di formule logiche proposizionali (requisiti su sistemi finiti)
  - ….
- Caratteristica che accomuna la soluzione di tutti questi problemi è che è "difficile" *trovare* la soluzione ma è facile *verificare* se una possibile soluzione lo è effettivamente:
  - se un diavoletto mi dice "prova questa", verificare se la "soffiata" è giusta o no non è difficile ---->
  - tipici problemi risolti in maniera esaustiva: le "provo tutte"
  - in modo nondeterministico: scelgo (ND) una possibile soluzione; verifico se lo è.
  - Ovviamente passando alla versione deterministica, provarle tutte diventa molto oneroso (si ricordi la visita degli alberi)
- Se ne ricava una -grandissima- classe di problemi (contenente gli esempi di sopra e decine di migliaia di altri problemi):

- NP: la classe dei problemi risolvibili nondeterministicamente in tempo polinomiale
- P: la classe dei problemi risolvibili deterministicamente in tempo polinomiale (i problemi trattabili)
- La grande domanda: P = NP??
- *Molto probabilmente* no! Però …
- Se P = NP potremmo risolvere in maniera "efficiente" un'enorme quantità di problemi oggi intrattabili o affrontati con euristiche, casi particolari, ecc.
- Il concetto di (NP) completezza: un "rappresentante" della classe racchiude in sé l'essenza di tutti i problemi della classe: troviamo la soluzione per esso e l'abbiamo trovata per tutti!
- Il bello è che nell'enorme congerie di NP, una grandissima quantità di problemi è a sua volta anche NP-completa: basterebbe risolverne uno in tempo polinomiale (deterministicamente) e P sarebbe = NP; basterebbe provare per uno solo di essi che è intrattabile e tutti gli altri lo sarebbero pure!
- Infine: nondeterminismo non è sinonimo di casualità però … le affascinanti prospettive della computazione probabilistica.