Complessità del calcolo

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

21 luglio 2017

La complessità del calcolo

Quanto efficientemente risolviamo un problema?

- Razionalizzazione dei concetti di:
  - *costo* di una computazione
  - misura del costo della computazione
- Primo passo: costruire strumenti per valutare la complessità di un calcolo
- Secondo passo: analisi di algoritmi e strutture dati notevoli (= risolvono problemi che accadono spesso)
- Obiettivo: saper progettare e combinare algoritmi e strutture dati per realizzare soluzioni *efficienti* (oltre che corrette)

La complessità del calcolo

Quanto efficientemente risolviamo un problema?

- Posto che un dato problema sia calcolabile, ci domandiamo: a che costo?
- Effettueremo analisi *quantitative* del(lo):
  - Tempo di calcolo impiegato
  - Spazio occupato (registri, caches, RAM, disco, nastro)
- Analisi fatta su criteri di costo oggettivi (e formalizzabili): non teniamo conto di costi di sviluppo, tradeoff ambientali tra obiettivi contrastanti...

Oltre la risolvibilità

Dipendenza dal formalismo di calcolo

- Per la tesi di Church-Turing, un problema è calcolabile o meno indipendentemente dallo strumento usato (purchè 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 $\exists z \in L_\tau = \{x \dagger y | y = \tau(x)\}$ può essere molto meno efficiente del calcolare la traduzione in qualche caso.
- É verosimile assumere che cambiando modello di calcolo non cambi il tempo di esecuzione? No.

Costruire uno strumento generale

Costo di calcolo “indipendente” dal formalismo

- Non abbiamo una “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 MT *deterministiche*

Complessità temporale e spaziale

Complessità temporale

- Data la computazione $c_0 \vdash^* c_r$ di $\mathcal{M}$ la complessità temporale è $T_{\mathcal{M}}(x) = r$ se $\mathcal{M}$ termina in c_r, ∞ altrimenti
- $\mathcal{M}$ è *deterministica* $\Rightarrow$ computazione unica sull'ingresso x

Complessità spaziale

- Data la computazione $c_0 \vdash^* c_r$ di $\mathcal{M}$ la complessità spaziale è $S_{\mathcal{M}}(x) = \sum_{j=1}^{r} \max_{i \in \{0,\dots,r\}}(|\alpha_{ij}|)$ con $|\alpha_{ij}|$ lunghezza del contenuto del j-esimo nastro alla mossa i-esima
  - Intuitivamente: la somma delle quantità massime di nastro occupate, per ogni nastro
- NB: $\forall x, \exists k \in \mathbb{N}, \frac{S_{\mathcal{M}}(x)}{k} \leq T_{\mathcal{M}}(x)$

Esempio: riconoscere $L = \{wcw^R\}$ con MT a 1 nastro

Complessità temporale

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

Complessità spaziale

- Se $x \in L$, $S_{\mathcal{M}}(x) = \left\lfloor \frac{|x|}{2} \right\rfloor$
- Se $x \notin L, x \in \{a, b\}^*$, $S_{\mathcal{M}}(x) = |x| + 1$
- Se $x \in c^+$, $S_{\mathcal{M}}(x) = 0$

Riducendo all'essenziale

Analisi su MT

- Contempla tutti i dettagli del funzionamento della MT (a 1 nastro nel caso precedente)
  - La complessità temporale dipende dal valore dell'input

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: Si tratta di una semplificazione: 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 $T_{\mathcal{M}}(.)$ che per $S_{\mathcal{M}}(.)$)

- Caso pessimo: $T_{\mathcal{M}}(n) = \max_{x,|x|=n} T_{\mathcal{M}}(x)$
- Caso ottimo: $T_{\mathcal{M}}(n) = \min_{x,|x|=n} T_{\mathcal{M}}(x)$
- Caso medio: $T_{\mathcal{M}}(n) = \frac{\sum_{x,|x|=n} T_{\mathcal{M}}(x)}{\mathbf{I}^n}$ (assumendo gli input uniformemente distribuiti)

Scelte tipiche

- Quasi sempre il caso pessimo:
  - É quello più rilevante (quasi sempre)
  - L'analisi risulta più semplice del caso medio

Tasso di crescita

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

- Sapere i valori esatti di $T_{\mathcal{M}}(n)$ e $S_{\mathcal{M}}(n)$ per un dato n è utile fino ad un certo punto
  - Dovremmo calcolarli per ogni n di nostro interesse!
- Focalizziamoci su quanto $T_{\mathcal{M}}(n)$ e $S_{\mathcal{M}}(n)$ crescono in funzione di n
- Osserviamo quindi $T_{\mathcal{M}}(n)$ e $S_{\mathcal{M}}(n)$ quando $n \to \infty$
  - Non distingueremo quindi $T_{\mathcal{M}}(n) = n^2 + 13n$ da $3n^2$, hanno comportamenti "simili"
- 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

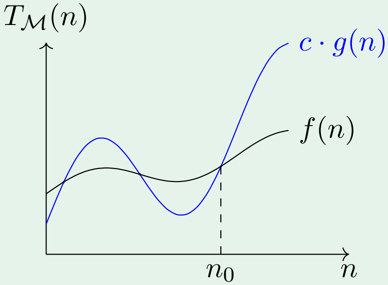
- Introduciamo una notazione per indicare il comportamento asintotico di una funzione:
  - La notazione $\mathcal{O}$-grande: limite asintotico superiore
  - La notazione Ω-grande: limite asintotico inferiore
  - La notazione Θ-grande: limite asintotico sia sup. che inf.
- Caveat pratico 1: comportamento asintotico $\rightarrow$ per valori piccoli di n potrebbe non essere un buon modello
- Caveat pratico 2: in qualche (raro) caso valore "piccolo" non corrisponde alla nostra intuizione
  - Un algoritmo con complessità asintotica maggiore può essere più lento di uno a complessità minore per valori piccoli di n
  - Tutto funziona ... fin quando non crescono i dati in ingresso

Notazione $\mathcal{O}$-grande

Definizione

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

Graficamente



- Le funzioni in $\mathcal{O}(g(n))$ dominano $f(n)$ a partire da n_0

Esempi

Facilmente

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

Leggermente meno facilmente

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

Un abuso comune

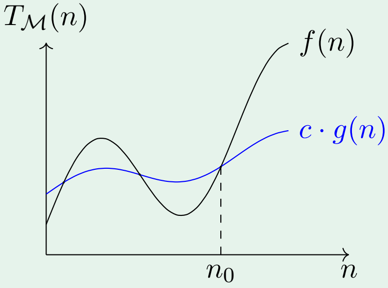
- É comune scrivere $f(n) = \mathcal{O}(g(n))$ al posto di $f(n) \in \mathcal{O}(g(n))$ (oppure $f(n)$ è $\mathcal{O}(g(n))$)

Notazione Ω-grande

Definizione

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

Graficamente



- Le funzioni in $\Omega(g(n))$ sono dominate da $f(n)$ 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 $5n^4 + 3 \notin \Omega(n^4 log(n))$
- $n2^n + n^{13} \in \Omega(2^n)$, ma
- $n2^n + n^{13} \in \Omega(n^{70})$

Proprietà

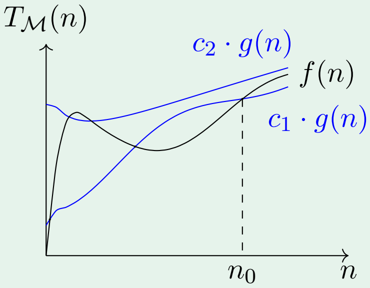
- Se e solo se $f(n) \in \mathcal{O}(g(n))$ allora $g(n) \in \Omega(f(n))$

Notazione Θ-grande

Definizione

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

Graficamente



- Le funzioni in $\Theta(f(n))$ hanno lo stesso andamento di $f(n)$ 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 di Θ

- Θ è una relazione di equivalenza sull'insieme di funzioni
  - Riflessiva: $f(n) \in \Theta(f(n))$
  - Simmetrica: $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$
  - Transitiva:
    $f(n) \in \Theta(g(n)) \wedge h(n) \in \Theta(g(n)) \Rightarrow f(n) \in \Theta(h(n))$

Proprietà di $\mathcal{O}, \Omega$

- Esse sono riflessive: $f(n) \in \mathcal{O}(f(n))$, $f(n) \in \Omega(f(n))$
- $f(n) \in \Theta(f(n))$ se e solo se
  $f(n) \in \mathcal{O}(f(n)) \wedge f(n) \in \Omega(f(n))$

Confronti operativi

Definizioni come limiti

- Se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c, c \neq 0, c \neq \infty$ allora $f(n) \in \Theta(g(n))$
  - Gli andamenti asintotici di f e g differiscono per una costante moltiplicativa
- Se $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ 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, cresce più velocemente)
  - Si indica anche, con notazione più sintetica, $\Theta(f(n)) < \Theta(g(n))$
- L'uso della notazione $\mathcal{O}, \Omega, \Theta$ estrae la porzione "più importante" di una funzione di complessità

Una prima applicazione

Riconoscere $L = \{wcw^r\}$ con MT 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 MT a 2 nastri:
  - Memorizzo solo la posizione 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 MT a 2 nastri $\mathtt{N_a}, \mathtt{N_b}$

1. Ciclo per trovare c, ad ogni passo incrementa il contatore memorizzato $\mathtt{N_a}$ [$T_{\mathcal{M}}(n)= n\log(n)$, $S_{\mathcal{M}}(n)=\log(n)$]
2. Ripeti fino a quando il contatore su $\mathtt{N_a}$ è $= 0$
   1. Copia il contenuto di $\mathtt{N_a}$ su $\mathtt{N_b}$ [$T_{\mathcal{M}}(n)=S_{\mathcal{M}}(n)=\log(n)$]
   2. Sposta la testina di ingresso di $\mathtt{N_b}$ posti a sx, decrementando $\mathtt{N_b}$ [$T_{\mathcal{M}}(n)= i\log(i)$, $S_{\mathcal{M}}(n)=\log(n)$]
   3. Memorizza il carattere nello stato [$T_{\mathcal{M}}(n)=S_{\mathcal{M}}(n)=k$]
   4. Ritorna al carattere c [$T_{\mathcal{M}}(n)=i$,$S_{\mathcal{M}}(n)=k$]
   5. Copia il contenuto di $\mathtt{N_a}$ su $\mathtt{N_b}$ [$T_{\mathcal{M}}(n)=S_{\mathcal{M}}(n)=\log(n)$]
   6. Sposta la testina di ingresso di $\mathtt{N_b}$ posti a dx, decrementando $\mathtt{N_b}$ [$T_{\mathcal{M}}(n)= i\log(i)$, $S_{\mathcal{M}}(n)=\log(n)$]
   7. Confronta il carattere con lo stato, se diversi halt [$T_{\mathcal{M}}(n)=S_{\mathcal{M}}(n)=k$]
   8. Decrementa $\mathtt{N_a}$ e torna a c [$T_{\mathcal{M}}(n)= \log(i) + i$, $S_{\mathcal{M}}(n)=\log(n)$]

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 $T_{\mathcal{M}}(n) \in \Theta(n^2 \log(n))$
- É un caso di compromesso spazio-temporale: la soluzione è più lenta ma occupa meno spazio
  - É possibile spesso precalcolare alcuni risultati utilizzati spesso e andare a leggerli da memoria
- Questo esempio giustifica perchè in una MT 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 MT ...
  - ... ma previene esempi (non banali) di algoritmi con $S_{\mathcal{M}}(n)$ sub-lineare

Modelli di calcolo

Altri modelli *deterministici* di calcolo

- FSA: hanno sempre $S_{\mathcal{FSA}}(n) \in \Theta(1)$, $T_{\mathcal{FSA}}(n) \in \Theta(n)$
  - Tecnicamente, $T_{\mathcal{FSA}}(n) = n$, leggono un carattere per mossa
- APD: hanno sempre $S_{\mathcal{APD}}(n) \in \mathcal{O}(n)$, $T_{\mathcal{APD}}(n) \in \Theta(n)$
- MT a nastro singolo?
  - É 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 $\Theta(n)$
  - Esiste un algoritmo più efficiente di $T_{\mathcal{M}}(n) \in \Theta(n^2)$ (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: MT a nastro singolo sono più potenti degli APD, ma ciò che eseguono può avere "qualunque" complessità spazio/temporale!

I teoremi di accelerazione lineare

Teorema

Se L *è accettato da una MT* $\mathcal{M}$ *a* k *nastri in* $S_{\mathcal{M}}(n)$*, per ogni* $c \in \mathbb{R}, c>0$ *posso costruire una MT* $\mathcal{M}'$ *a* k *nastri 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 Γ_i dell'i-esimo nastro di $\mathcal{M}$ costruisco Γ_i' di $\mathcal{M}'$ assegnando un elemento per ogni $s \in \Gamma_i^r$
- Costruisco l'OC 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 ong r movimenti di $\mathcal{M}$
  - Memorizzi la posizione della testina "all'interno" dei nuovi simboli degli alfabeti di nastro Γ_i

I teoremi di accelerazione lineare - 2

Teorema

Se L è accettato da una MT $\mathcal{M}$ a k nastri in $S_{\mathcal{M}}(n)$, posso costruire una MT $\mathcal{M}'$ a 1 nastro (non nastro singolo) 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 MT $\mathcal{M}$ a k nastri in $S_{\mathcal{M}}(n)$, per ogni $c \in \mathbb{R}, c{>}0$ posso costruire una MT $\mathcal{M}'$ a 1 nastri 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 MT $\mathcal{M}$ a k nastri in $T_{\mathcal{M}}(n)$, per ogni $c \in \mathbb{R}, c>0$ posso costruire una MT $\mathcal{M}'$ a $k+1$ nastri 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 MT vale anche per i calcolatori a-la Von Neumann
  - Equivale a cambiare la dimensione della word della macchina ($32\text{b} \rightarrow 64\text{b}$) o, equivalentemente, ad usare operazioni vettoriali (MMX/SSEn)
- Possiamo avere speedup *lineari* arbitrariamente grandi, aumentando il parallelsimo 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

MT 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), la MT necessita 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 MT 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

Un calcolatore semplificato

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

La macchina RAM

Instruction set e semantica pseudo-RTL

Istruzione	Semantica
`LOAD X`	$\mathtt{M}[0] \leftarrow \mathtt{M}[X]$
`LOAD= X`	$\mathtt{M}[0] \leftarrow X$
`LOAD* X`	$\mathtt{M}[0] \leftarrow \mathtt{M}[\mathtt{M}[X]]$
`STORE X`	$\mathtt{M}[X] \leftarrow \mathtt{M}[0]$
`STORE* X`	$\mathtt{M}[\mathtt{M}[X]] \leftarrow \mathtt{M}[0]$
`ADD X`	$\mathtt{M}[0] \leftarrow \mathtt{M}[0] + \mathtt{M}[X]$
`SUB X`	$\mathtt{M}[0] \leftarrow \mathtt{M}[0] - \mathtt{M}[X]$
`MUL X`	$\mathtt{M}[0] \leftarrow \mathtt{M}[0] \times \mathtt{M}[X]$
`DIV X`	$\mathtt{M}[0] \leftarrow \mathtt{M}[0] / \mathtt{M}[X]$
`HALT`	–

Istruzione	Semantica
`READ X`	$\mathtt{M}[X] \leftarrow \mathtt{In}$
`READ* X`	$\mathtt{M}[\mathtt{M}[X]] \leftarrow \mathtt{In}$
`WRITE X`	$\mathtt{Out} \leftarrow \mathtt{M}[X]$
`WRITE= X`	$\mathtt{Out} \leftarrow X$
`WRITE* X`	$\mathtt{Out} \leftarrow \mathtt{M}[\mathtt{M}[X]]$
`JUMP l`	$\mathtt{PC} \leftarrow l$
`JZ l`	Se $\mathtt{M}[0] = 0$ $\mathtt{PC} \leftarrow l$

Ci sono `JGT,JGZ,JLT,JLZ`

Test (semplice) di primalità in assembly RAM

Stampa 1 se l'intero su nastro è primo 0 altrimenti

```
         READ 1
         LOAD= 1
         SUB 1
         JZ no
         LOAD= 2
         STORE 2
loop:    LOAD 1
         SUB 2
         JZ yes
         LOAD 1
         DIV 2
         MUL 2
         SUB 1
         JZ no
         LOAD 2
         ADD= 1
         STORE 2
         JUMP loop
yes:     WRITE= 1
         HALT
no:      WRITE= 0
         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_2
- Le istruzioni 7 – 18 hanno costo costante k_3, ma sono eseguite, nel caso pessimo n volte
- $T_{\mathcal{RAM}}(n) = k_1 + nk_2 + k_3 = \Theta(n)$
- $S_{\mathcal{RAM}}(n) = 3 = \Theta(1)$ (uso solo 3 celle di mem)
- N.B.: questa volta n non è la lunghezza dell'ingresso!

Analisi di altri algoritmi

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

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

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
- Analizziamo la porzione di codice dopo che:
  - La sequenza è stata caricata in memoria
  - $\mathtt{M}[1]$ e $\mathtt{M}[2]$ contengono gli indirizzi della prima e dell'ultima cella della sequenza stessa

Ricerca Binaria: codice

```
        READ 3
        LOAD 1
        STORE 4
        ADD 2
        SUB= 1
        STORE 5
loop:   LOAD 5
        SUB 4
        JZ no
        LOAD 5
        ADD 4
        DIV= 2
        STORE 6
        LOAD* 6
        SUB 3
        JZ yes
        JLT low
        JUMP high
low:    LOAD 6
        SUB= 1
        STORE 5
        JUMP loop
high:   LOAD 6
        ADD= 1
        STORE 4
        JUMP loop
yes:    WRITE= 1
        HALT
no:     WRITE= 0
        HALT
```

Costo complessivo: $T_{\mathcal{RAM}}(n) = \Theta(\log(n))$

Limiti del criterio di costo

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(n);
```

- Che complessità temporale ha l'implementazione qui sopra?
  $T_{\mathcal{RAM}}(n) = k_1 + nk_2 + k_3 = \Theta(n)$
- 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
  - É 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
  - Anche gli indirizzi di memoria sono interi $\rightarrow$ stesse problematiche nel (raro) caso in cui parola della macchina non riesce a contenere il massimo indirizzo (μC a 8 bit)

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: metodo scolastico $\Theta(d^2)$
    - Primo miglioramento: $\Theta(d^{\log_2(3)}) \approx \Theta(d^{1.58})$
    - Ulteriore miglioramento: $\Theta(d \log(d) \log(\log(d)))$
    - Miglior algoritmo attuale: $\Theta(d \log(d)\ 2^{\mathcal{O}(\log^*(d))})$
  - Divisioni: metodo scolastico $\Theta(d^2)$, o al costo dell'algoritmo di moltiplicazione scelto
- `JUMP` e `HALT` sono a costo costante, così come le `Jcc` (il valore del codice di condizione è nella PSW, già calcolato)

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

Analisi di caso pessimo, `MUL` scolastica

	`READ 2`	$\log(n)$
	`LOAD= 2`	$\log(2) = k$
	`STORE 1`	$\log(2) = k$
`loop:`	`LOAD 1`	$\log(2^{2^{n-1}}) = 2^{n-1}$
	`MUL 1`	$(\log(2^{2^{n-1}}))^2 = (2^{n-1})^2 = 2^{2n-2}$
	`STORE 1`	$\log(2^{2^n}) = 2^n$
	`LOAD 2`	$\log(n)$
	`SUB= 1`	$\log(n)$
	`STORE 2`	$\log(n-1)$
	`JGT loop`	1
	`WRITE 1`	$\log(2^{2^n}) = 2^n$
	`HALT`	1

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

Rapporti tra criteri di costo

Ri-analizzando

- Riconoscere $L = \{wcw^R\}$ è $\Theta(n \log(N))$ (colpa del contatore)
- Ricerca binaria: $\Theta(\log(n)^2)$ (colpa degli indici)

Quale criterio scegliere?

- Se la dimensione di ogni singolo elemento in ingresso non varia significativamente nell'esecuzione dell'algoritmo (= stesso numero di cifre per tutta l'esecuzione): costo costante
- Nel caso in cui ci sia un significativo cambio nella dimensione dei singoli elementi in ingresso (= il numero di cifre cresce in modo significativo): costo logaritmico

Rapporti tra complessità con diversi modelli di calcolo

Modelli di calcolo diversi $\rightarrow$ diversa efficienza

- Risolvere lo stesso problema con macchine diverse può dare luogo a complessità diverse
- Non esiste un modello migliore in assoluto

Tesi di correlazione polinomiale

- Sotto "ragionevoli" ipotesi di criteri di costo, se un problema è risolvibile da $\mathcal{M}$ in $T_{\mathcal{M}}(n)$, allora è risolvibile da un qualsiasi altro modello (Turing-completo) $\mathcal{M}'$ in $\pi(T_{\mathcal{M}'}(n))$ dove $\pi(\cdot)$ è un opportuno polinomio
- Dimostriamo il teorema di correlazione (temporale) polinomiale tra MT e RAM

Correlazione temporale tra MT a k nastri e RAM

RAM simula MT a k nastri: simulazione delle azioni

- Mappiamo la MT sulla RAM:
  - Stato della MT $\rightarrow$ 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 MT a k nastri e RAM

RAM simula MT a k nastri: lettura

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

RAM simula MT 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 MT con un k di mosse: $T_{\mathcal{RAM}}(n) = \Theta(T_{\mathcal{M}}(n))$ ($= \Theta(T_{\mathcal{M}}(n)\log(T_{\mathcal{M}}(n)))$ costo log.)

Correlazione temporale tra MT a k nastri e RAM

MT a k nastri simula RAM (senza MUL/DIV per semplicità)

- Organizziamo un nastro della MT così:

$\dots$	$\triangleleft$	i_j	$\ddagger$	$\mathtt{M}[i_j]$	$\triangleright$	$\dots$	$\triangleleft$	i_k	$\ddagger$	$\mathtt{M}[i_k]$	$\triangleright$	$\dots$

- Il nastro è inizialmente vuoto: salviamo solo le celle in cui è avvenuta una STORE
- Usiamo un ulteriore nastro per contenere $\mathtt{M}[0]$ in binario
- Usiamo un ultimo nastro per consentirci di spostare dati su un punto principale quando dobbiamo salvare per la prima volta $\mathtt{M}[i_j]$ ma $\mathtt{M}[i_k]$ e $\mathtt{M}[i_l]$, $i_k < i_j < i_l$, sono state già salvate

Correlazione temporale tra MT a k nastri e RAM

MT a k nastri simula RAM: simulare istruzioni

- `LOAD x`: cerco x sul nastro principale, copio la porzione accanto nella zona dati di $\texttt{M}[0]$ usando il nastro di supporto
- `STORE x`: cerco x sul nastro principale:
  - Se lo trovo, salvo il valore di $\texttt{M}[0]$
  - Altrimenti, creo dello spazio usando il nastro di servizio se necessario e salvo
- `ADD x`: cerco x, copio $\texttt{M}[x]$ sul nastro di supporto, calcolo la somma scrivendo direttamente in $\texttt{M}[0]$
- In generale: simulare una mossa della RAM richiede alla MT un numero di mosse $\leq$ da $c\,*$ (lunghezza del nastro principale)

Correlazione temporale tra MT a k nastri e RAM

Lemma (Occupazione sul nastro principale)

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

Dimostrazione.

- Ogni cella della RAM occupa $\log(i_j) + \log(\mathtt{M}[i_j])$
- Ogni cella della RAM viene materializzata solo se la RAM effettua una `STORE`
- La `STORE` costa alla RAM $\log(i_j) + \log(\mathtt{M}[i_j])$
- Per riempire r celle la RAM ci mette $\sum_{j=1}^{r} \log(i_j) + \log(\mathtt{M}[i_j])$ in tempo: quantità identica allo spazio che occupano sul nastro della MT ■

Correlazione temporale tra MT a k nastri e RAM

Concludendo

- La MT impiega al più $\Theta(T_{\mathcal{RAM}}(n))$ per simulare una mossa della RAM
- Se la RAM ha complessità $T_{\mathcal{RAM}}(n)$ essa effettua al più $T_{\mathcal{RAM}}(n)$ mosse (ogni mossa costa almeno 1)
- La simulazione completa della RAM da parte della MT costa al più $\Theta((T_{\mathcal{RAM}}(n))^2)$ il legame tra $T_{\mathcal{RAM}}(n)$ e $T_{\mathcal{MT}}(n)$ tra è polinomiale

Conseguenze della correlazione temporale tra MT RAM

Implicazioni della correlazione polinomiale

- La relazione polinomiale tra il tempo di calcolo su $T_{\mathcal{RAM}}(n)$ e $T_{\mathcal{MT}}(n)$ ci consente di definire la classe di problemi risolvibili in tempo/spazio polinomiale (classe P)
- Questo risultato ha portato alla formulazione di una "tesi di trattabilità": i problemi risolvibili in P sono quelli trattabili
- La classe P comprende anche polinomi come n^{30}, ma l'esperienza pratica conferma che la maggioranza dei problemi polinomiali di interesse pratico ha anche un grado del polinomio accettabile