LABORATORIO DI PROGRAMMAZIONE
Corso di laurea in matematica

12 – LA COMPLESSITA' COMPUTAZIONALE

Marco Lapegna
Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

wpage.unina.it/lapegna

Dalle lezioni precedenti

Principali strumenti per lo sviluppo di algoritmi in pascal-like

- Variabili e costanti
- Variabili strutturate
  - array
- Strutture di controllo
  - Strutture di selezione
  - Strutture di iterazione
- Procedure

concetti universali presenti in tutti i linguaggi di programmazione

Qualunque algoritmo puo' essere ricondotto ad un algoritmo equivalente basato solo su tali costrutti (teorema di Bohm-Jacopini, 1966)

Problema

Siano:

- P un problema da risolvere
- A un algoritmo utilizzato per risolvere P
- Sia C un calcolatore utilizzato per eseguire A
- Sia T il tempo impiegato per risolvere il problema P eseguendo l'algoritmo A sul calcolatore C

PROBLEMA

Come ridurre il tempo di esecuzione T?

Soluzione 1:

Utilizzare un calcolatore migliore (piu' veloce)

Soluzione 2:

Utilizzare un algoritmo migliore (piu' efficiente)

OBIETTIVO:

Determinare una metodologia per valutare gli algoritmi indipendentemente dall'ambiente di calcolo

Esempio

Dato un numero reale ***X*** ,
si voglia calcolare ***Y = X¹⁶***

e' possibile utilizzare uno dei due algoritmi:

Algoritmo 1:

$X^2 = X * X$

$X^3 = X^2 * X$

...

$Y = X^{16} = X^{15} * X$

15 moltiplicazioni

Algoritmo 2:

$X^2 = X*X$

$X^4 = X^2 * X^2$

$X^8 = X^4 * X^4$

$Y = X^{16} = X^8 * X^8$

4 moltiplicazioni

```
Begin Algoritmo1
var X, Y : real
var i, N: integer

read X, N
Y = X
for i=1 to N-1
      Y = Y * X
endfor

Print Y
end Algoritmo1
```

Algoritmo 1

```
Begin Algoritmo2
var X, Y : real
var i, N: integer

read X, N
Y = X
for i=1 to log(N)
      Y = Y * Y
endfor

Print Y
end Algoritmo2
```

Algoritmo 2

In generale

Per calcolare $Y = X^N$
(N potenza di 2)

Algoritmo 1
richiede **N-1 operazioni**
moltiplicando X per se stesso ripetutamente

Algoritmo 2
richiede **log(N) operazioni**
elevando al quadrato ripetutamente

Il secondo algoritmo e' piu' efficiente

L'efficienza di un algoritmo dipende dal **numero di operazioni** richieste per ottenere la soluzione

N		Algoritmo 1	Algoritmo 2
8 =	2^3	7	3
32 =	2^5	31	5
256 =	2^8	255	8
1024 =	2^{10}	1023	10
8192 =	2^{13}	8191	13
65536 =	2^{16}	65535	16
524288 =	2^{19}	524287	19

Confronto tra il numero di operazioni dell'Algoritmo 1 e quello dell'Algoritmo 2 al crescere della dimensione del problema N

Problema

Molte applicazioni
richiedono l'utilizzo di matrici

- Di **grandi dimensioni**
- **Sparse** (con molti zeri)

Esempio: matrice A di ordine N=8

Soluzione 1:
Memorizzazione mediante **array bidimensionale**

var: A(8,8) array of real

Sono necessarie 64 parole in memoria

$$A = \begin{pmatrix} 7 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -6 \\ -3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ -1 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 8 & 0 & 3 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Memorizzazione di una matrice sparsa di ordine N=8 mediante un array bidimensionale

Soluzione 2

Utilizzando una **forma compatta memorizzando solo i valori diversi da zero**

Sono sufficienti 3 array

- **AC** contiene gli elementi non nulli di A riga per riga
- **J** contiene l'indice di colonna degli elementi di A riportati in AC
- **I** contiene la posizione in AC del primo elemento di ogni riga di A

Il secondo schema richiede
15+15+9=39 parole in memoria

Il secondo algoritmo e' piu' efficiente

L'efficienza di un algoritmo dipende dallo **spazio** richiesto per memorizzare i dati

AC =	7	-1	2	8	1	-6	-3	2	4	-1	7	9	8	3	2
J=	1	4	1	2	5	8	1	7	6	1	4	1	5	7	3
I=	1	3	5	7	9	10	12	15	16						

Memorizzazione di una matrice sparsa di ordine N=8 in forma compatta

Complessita' computazionale di un algoritmo

Per valutare l'efficienza di un algoritmo si misura il
Costo Computazionale dell'algoritmo, cioe'

- **Il numero di operazioni effettuate dall'algoritmo**
- **Lo spazio di memoria richiesto dai dati**

Detta N la dimensione del problema,
per misurare il costo computazionale di un algoritmo si definiscono

- **Una funzione complessita' di tempo T(N)**
- **Una funzione complessita' di spazio S(N)**

Esempio

Valutare la complessita' di tempo T(N) per la costruzione del Triangolo di Tartaglia di N righe

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
1   6  15  20  15   6   1
```

- In ciascuna riga, ogni elemento e' la somma dell'elemento superiore e di quello in alto a sinistra
- Se l'elemento manca si sottintende 0

```
Begin tartaglia
var TAR(10,10):  array of real
var i, j,  N: integer

read  N

TAR(1,1) = 1
for i=2 to N
      TAR(i,1) = 1
      for j = 2 to i -1
          TAR(i,j) = TAR(i-1,j) + TAR(i-1,j-1)
      endfor
      TAR(i,i) = 1
endfor

for  i = 1 to N
       for j = 1 to i
           print TAR(i,j)
       endfor
endfor

end tartaglia
```

Algoritmo per il triangolo di Tartaglia

Complessita' di tempo

Numero di operazioni dell' algoritmo

- Prima riga e seconda riga: 0 somme
- Terza riga: 1 somma
- …
- N-ma riga: N-2 somme

Totale:

$T(N) = 1 + 2 + \ldots + N-2$ addizioni

$T(N) = (N-1)(N-2) / 2$ addizioni

$= (N^2 - 3N + 2)/2$

Al crescere di N
il termine dominante e' N^2

Si studia il comportamento di T(N) e S(N) al crescere di N (complessita' asintotica)

N	T(N)
2	(4-2)/2
10	(100-28)/2
100	(10000-298)/2
1000	(1000000-2998)/2
10000	$(10^8 + 3\ 10^4 + 2)/2$

Valori di T(N) per differenti valori di N

Notazione " O grande"

Definizione

Data una funzione positiva di riferimento *f(N)* si dice che

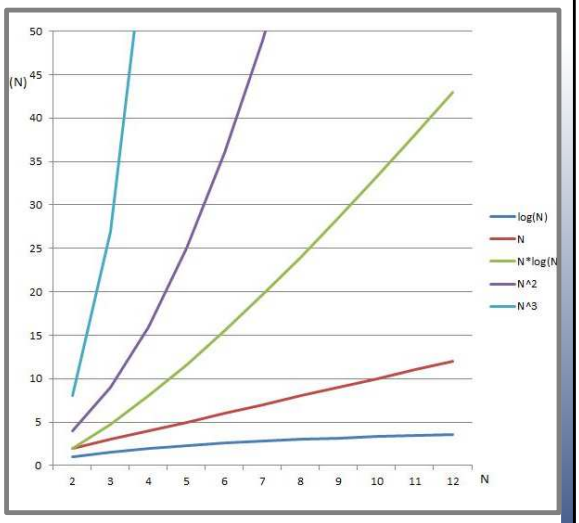
$T(N) \in O(f(N)) \quad N \Rightarrow \infty$

se e solo se

$\exists \ N_0$, M tale che

$T(N) \leq M\ f(N)$ per ogni $N > N_0$

f(N) fornisce un **limite superiore** asintotico dell'andamento di T(N)



Andamento delle piu' comuni funzioni di riferimento per la complessita' computazionale

Osservazione

In molti casi, la complessita' di tempo T(N) di uno stesso algoritmo puo' variare a secondo dei dati

ESEMPIO:

Ricerca di un nome in un elenco:

Caso peggiore : nome non presente
Si scorre tutto l'elenco (T(N) = N confronti)

Caso migliore : nome presente nella prima componente
Si termina subito (T(N) = 1 solo confronto)

Caso peggiore e caso migliore nella ricerca sequenziale

Notazione "Omega grande"

Definizione

Data una funzione positiva di riferimento *g(N)* si dice che
$T(N) \in \Omega(g(N))$ $N \Rightarrow \infty$

se e solo se

$\exists$ N_0 , M tale che

$T(N) \geq M\, g(N)$ per ogni $N > N_0$

g(N) fornisce un **limite inferiore**
asintotico dell'andamento di T(N)

riassumendo

Parametri per la valutazione della complessita' computazionale degli algoritmi

- Complessita' di tempo T(N): numero di operazioni
- Complessita' di spazio S(N): numero di parole occupate dai dati in memoria

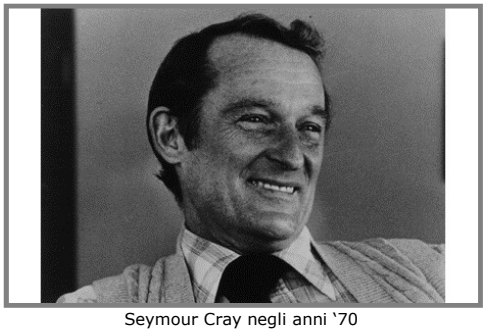
Complessita' asintotica

- *Notazione "O grande"* T(N) O(f(N)) fornisce un limite superiore asintotico dell'andamento di T(N). Indicativo del "Caso Peggiore" (worst case)
- Notazione "Omega grande" T(N) Ω(g(N)) fornisce un limite inferiore asintotico dell'andamento di T(N). Indicativo del "Caso Migliore" (best case)

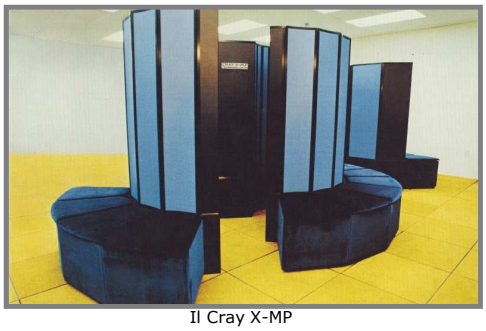
Un po' di storia (12)

Seymour Cray (1925 – 1996)

- Ingegnere elettronico americano, e' stato uno dei piu' geniali progettisti di supercalcolatori. Alla Control Data progetta negli anni '60 i piu' potenti calcolatori dell'epoca: il CDC 6600 e il CDC 7600
- Nel 1972 fondo' la Cray Research che sviluppa il Cray-1 nel 1976 (ancora il piu' potente calcolatore dell'epoca) e il Cray X-MP nel 1982: primo calcolatore parallelo (multiprocessore) della storia con 4 unita' di calcolo indipendenti.
- Il Cray X-MP aveva una potenza di picco di 800 Mflops ed una memoria di 128 MByte. Con il Cray X-MP inizia l'era del calcolo parallelo e del calcolo ad alte prestazioni.
- Seymour Cray muore in un incidente stradale, mentre la Cray Research, dopo alcune trasformazioni societarie, e' ancora oggi presente stabilmente nelle prime posizioni della Top500 list.



Seymour Cray negli anni '70
(courtesy of Computer History Museum)



Il Cray X-MP
(courtesy of Computer History Museum)