

Analisi asintotica

Vittorio Maniezzo - University of Bologna

1

Analisi asintotica

Obiettivo:

- **semplificare** l'analisi del consumo di risorse di un algoritmo prescindendo dai dettagli implementativi o di altro genere.
- **classificare** gli algoritmi in base al loro comportamento asintotico.

Astrazione: come il tempo di esecuzione cresce in funzione della taglia dell'input *asintoticamente*.

Asintoticamente implica **non per tutti gli input**. Esempio: non per input di piccole dimensioni.

Vittorio Maniezzo - University of Bologna

2

2

Costo di esecuzione

Definizione

Un algoritmo A ha **costo di esecuzione** $O(f(n))$

- rispetto ad una certa risorsa di calcolo
- su istanze di ingresso di dimensione n

se la quantità $r(n)$ di risorsa sufficiente per eseguire A **su una qualunque istanza di dimensione n** verifica la relazione $r(n)=O(f(n))$.

Le risorse di calcolo di interesse sono soprattutto **tempo di calcolo** e **occupazione di memoria**.

Esempio: sequenza di istruzioni

istruzione 1;

istruzione 2;

...

istruzione k;

Il tempo totale è dato dalla **somma dei tempi di ciascuna istruzione**:

$$t_{\text{totale}} = t(\text{istruzione 1}) + t(\text{istruzione 2}) + \dots + t(\text{istruzione k})$$

Se ogni istruzione è "semplice" (coinvolge solo operazioni che vengono tradotte in un numero fisso di istruzioni assembler) allora **il tempo di ogni istruzione è costante**, e lo è anche il tempo totale: $\Theta(1)$.

Nel seguito assumeremo sempre che le istruzioni siano semplici.

Blocchi if - then - else

```
if (condizione)
  then sequenza di istruzioni 1
  else sequenza di istruzioni 2
```

Viene eseguita o la sequenza 1 o la sequenza 2.

Il tempo nel caso pessimo è il maggiore dei due:

$$\max(t(\text{sequenza 1}), t(\text{sequenza 2}))$$

Per esempio, se la sequenza 1 fosse $\Theta(n)$ e la sequenza 2 fosse $\Theta(1)$, il tempo nel caso pessimo del blocco if-then-else sarebbe $\Theta(n)$.

Cicli for

```
for (i = 1 ... n)
  sequenza di istruzioni
```

Il ciclo viene eseguito n volte, quindi anche la sequenza di istruzioni viene eseguita n volte.

Assumendo che ogni istruzione sia $\Theta(1)$, il tempo totale del ciclo è $n \cdot \Theta(1)$, cioè $\Theta(n)$.

Costo: $\Theta(n)$

Cicli for annidati

Caso 1: cicli in cui il numero di ripetizioni del ciclo interno è indipendente dall'indice del ciclo esterno:

```
for (i = 1 ... n)
  for (j = 1 ... m)
    sequenza di istruzioni
```


- Il ciclo esterno viene eseguito n volte.
- Ad ogni iterazione esterna, il ciclo interno viene eseguito m volte.
- Le istruzioni dentro al ciclo interno vengono eseguite $n*m$ volte.

Un caso particolare si ha quando anche il ciclo interno è eseguito n volte, complessità $\Theta(n^2)$

Cicli for annidati

Caso 2: cicli in cui il numero di ripetizioni del ciclo interno è dipendente dall'indice del ciclo esterno:

```
for (i = 1 ... n)
  for (j = 1 ... i)
    sequenza di istruzioni
```



Costo: ?

Cicli for annidati

```
for (i = 1 ... n)
  for (j = 1 ... n)
    for (k = 1 ... n)
      sum[i][j] += entry[i][j][k];
```

Costo: ?

Cicli for annidati

```
for (i = 1 ... n)
  for (j = 1 ... n)
    sum[i] += entry[i][j][0];

for (i = 1 ... n)
  for (k = 1 ... n)
    sum[i] += entry[i][0][k];
```

Cicli for annidati

```
for (i = 1 ... n)
  for (j = 1 ... sqrt(n))
    sequenza di istruzioni
```

```
for (i = 1 ... n)
  for (j = 1 ... sqrt(995))
    sequenza di istruzioni
```


Cicli while

```
i = tot = 1
while (i < n)
  tot += i
  i = i * 2
endWhile
```

Costo: ?

Cicli while: equivalente?

```
i = tot = n;  
while (i > 0)  
    tot += i;  
    i = i / 2;  
endWhile
```



(*i* variabile intera)
Costo: ?

subroutines

```
procedure main  
    for (i = 1 ... n)  
        tot += total(i);
```

```
procedure total(k)  
    for(i = 1 ... k)  
        subtotal += i;  
    return subtotal
```

Costo: ?

Analisi di algoritmi non ricorsivi

Ricerca il valore minimo in un array $v[]$ non vuoto

```
procedure minimo (v[])  
  m = 1; // posizione dell'elemento minimo  
  for (i = 2 ... v.length)  
    if(v[i]<v[m]) then m = i  
  return v[m]
```

Analisi

Sia n la lunghezza del vettore v .

- Il corpo del ciclo viene eseguito $n-1$ volte;
- Ogni iterazione ha costo $O(1)$ (vengono eseguite solo istruzioni elementari).
- Il costo di esecuzione della funzione minimo rispetto al tempo è quindi $O(n)$ (o meglio, $\Theta(n)$).

Vittorio Maniezzo - University of Bologna

15

15

Esercizi: complessità



- Due cicli in fila:

```
for (i = 1 ... n)  
  sequenza di istruzioni  
for (j = 1 ... m)  
  sequenza di istruzioni
```

Quale sarebbe la complessità se il secondo ciclo fosse ripetuto n volte?

- Ciclo annidato seguito da uno non annidato:

```
for (i = 1 ... n)  
  for (j = 1 ... n)  
    sequenza di istruzioni  
for (k = 1 ... n)  
  sequenza di istruzioni
```

- Ciclo annidato in cui il ciclo interno dipende dall'indice di quello esterno:

```
for (i = 1 ... n)  
  for (j = i ... n)  
    sequenza di istruzioni
```

Vittorio Maniezzo - University of Bologna

16

16

Algoritmi ricorsivi

Risultati usando *equazioni ricorsive*

Un'equazione tempi di esecuzione di algoritmi ricorsivi possono essere descritti **ricorsiva** esprime il valore di $f(n)$ come combinazione di $f(n_1), \dots, f(n_k)$ dove $n_i < n$.

- In pratica, data una sequenza $a_1, a_2, a_3, \dots, a_n$, l'equazione ricorsiva richiede il calcolo di tutti i termini precedenti per poter calcolare a_n .
- È necessario *conoscere esplicitamente il valore del primo termine*.

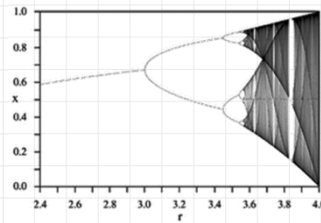
Esempi:

Mappa logistica: $x_{n+1} = rx_n(1 - x_n)$

Data la costante r e $0 < x_0 < 1$.

Numeri di Fibonacci

$$F_n = F_{n-1} + F_{n-2} \text{ con valori iniziali } F_0 = 0 \text{ e } F_1 = 1$$



Vittorio Maniezzo - University of Bologna

17

17

Un algoritmo ricorsivo

```
ricercaBinaria(val,v[],i,j)          Cerca l'elemento val
                                     in un array v ordinato
if(i>j)
then return -1      // non trovato
else
  m=(i+j)/2
  if (v[m]==val)
  then return m      // trovato
  else
    if(v[m]>val)
    then return RicercaBinaria(val,v,i,m-1)
    else return RicercaBinaria(val,v,m+1,j)
```

Vittorio Maniezzo - University of Bologna

18

18

Algoritmo ricorsivo

Analisi dell'algoritmo di ricerca binaria

Sia $T(n)$ il tempo di esecuzione della funzione **ricercaBinaria** su un vettore di $n=j-i+1$ elementi.

- In generale $T(n)$ dipende non solo dal numero di elementi su cui fare la ricerca, ma **anche dalla posizione** dell'elemento cercato (oppure dal fatto che l'elemento non sia presente).
- Nell'ipotesi più favorevole (**caso ottimo**) l'elemento cercato è proprio quello che occupa posizione centrale; in tal caso $T(n) = O(1)$.
- Nel caso meno favorevole (**caso pessimo**) l'elemento cercato non esiste.

Quanto vale $T(n)$ in questa situazione?

Vittorio Maniezzo - University of Bologna

19

19

Equazioni ricorsive: ricerca binaria

Analisi dell'algoritmo di ricerca binaria

Possiamo definire $T(n)$ per ricorrenza, come segue.

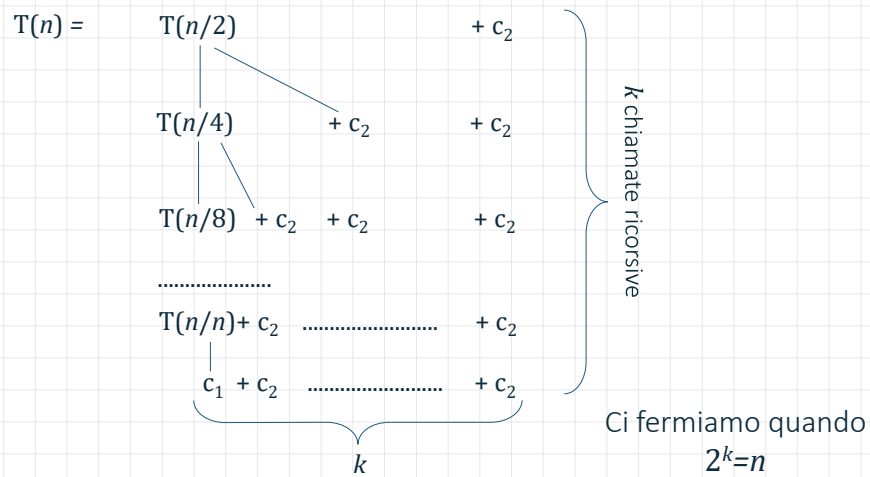
$$T(n) = \begin{cases} c_1 & \text{se } n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c_2 & \text{se } n > 1 \end{cases}$$

Vittorio Maniezzo - University of Bologna

20

20

soluzione: metodo iterativo



soluzione: metodo iterativo

Dobbiamo valutare k .

sappiamo che $2^k = n$, quindi

$\log_2(2^k) = \log_2(n)$, ovvero

$k = \log_2(n)$

2- Dimostrazione per induzione

Dobbiamo dimostrare che una affermazione è vera per ogni $n \geq 0$

Teorema

se

1. affermazione(0) è vera.
2. affermazione(n-1) vera implica affermazione(n) vera.

allora

affermazione(n) vera per ogni $n \geq 0$

Vittorio Maniezzo - University of Bologna

23

23

Es. dimostrazione per induzione:

Teorema

$$\text{affermazione}(n) = \sum_{i=1}^n i = n(n+1)/2$$

Ipotesi

$$\text{affermazione}(1) = \sum_{i=1}^1 i = 1(1+1)/2$$

Base

$$\text{affermazione}(n-1) \rightarrow \text{affermazione}(n):$$

Induttiva

$$\sum_{i=1}^{n-1} i = (n-1)(n)/2 \rightarrow \sum_{i=1}^n i = n(n+1)/2$$

$$\dots \text{ ma } \sum_{i=1}^n i = \sum_{i=1}^{n-1} i + n = (n-1)(n)/2 + n = n(n+1)/2$$

□

L'uguaglianza tra questi due termini è *affermazione(n-1)* che assumiamo vera per *ipotesi induttiva*.

Vittorio Maniezzo - University of Bologna

24

24

Metodo di sostituzione

Primo passo:

Ci buttiamo a “indovinare” una possibile soluzione:

$$T(n) \leq c \log_2(n)$$

Secondo passo:

La *verifichiamo* per induzione come segue:

Assumiamo che $T(n') \leq c \log_2(n')$ per $n' < n$

e dimostriamo che $T(n) \leq c \log_2(n)$

c è una costante (indipendente da n) che determineremo strada facendo

Metodo di sostituzione

$$T(n) = T(n/2) + 1$$

$$\leq c \log_2(n/2) + 1$$

$$= c \log_2(n) - c \log_2(2) + 1$$

$$= c \log_2(n) - c + 1$$

se $c \geq 1$ allora

$$\leq c \log_2(n)$$

Ipotesi induttiva !!! ($n/2$ è n')



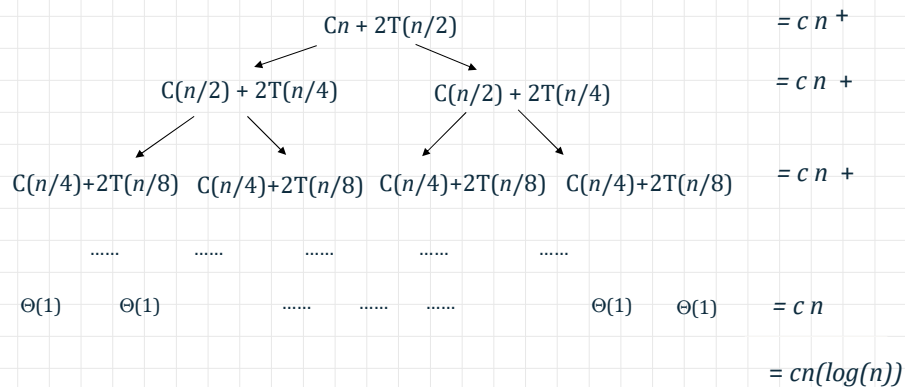
un esempio piú complicato

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Soluzione $T(n) = \Theta(n \log(n))$

27

Metodo iterativo



Il fattore $\log(n)$ deriva dal fatto che l'albero ha un'altezza $\log(n)$

28

"Master Method"

$$T(n) = aT(n/b) + f(n) \quad a \geq 1, b > 1, f(n) > 0$$

Poniamo $x = \log_b a$

$$f(n) = O(n^{x-\varepsilon}) \text{ con } \varepsilon > 0 \quad \Rightarrow \quad T(n) = \Theta(n^x)$$

$$f(n) = \Theta(n^x) \quad \Rightarrow \quad T(n) = \Theta(n^x \log(n))$$

$$\left\{ \begin{array}{l} f(n) = \Omega(n^x) \\ af(n/b) \leq cf(n) \\ \text{per tutti gli } n > n_0 \end{array} \right\} \quad \text{allora } T(n) = \Theta(f(n))$$

Non lo chiedo