

COMPLESSITÀ COMPUTAZIONALE DEGLI ALGORITMI

Fondamenti di Informatica a.a.2005/06
Prof. V.L. Plantamura
Dott.ssa A. Angelini

Ricorsività

- Si definisce ricorsiva una procedura P che invoca, direttamente o indirettamente, se stessa.
- Tali procedure sono facili da comprendere e analizzare:
 - La correttezza di un programma ricorsivo si dimostra facilmente con il metodo induttivo;
 - Il calcolo della complessità temporale si riduce alla soluzione di una relazione di ricorrenza.

Fondamenti di Informatica a.a. 2005/06

Fattoriale – Metodo iterativo

- Calcolo del fattoriale di un numero,
 $n! = n \cdot (n-1) \cdot \dots \cdot 1$

```
int Fattoriale(int n) {  
    int fatt;  
    fatt = 1;  
    for (i=0; i <= n; i++) {  
        fatt = fatt * i;  
    }  
    return(fatt); }
```

Fondamenti di Informatica a.a. 2005/06

Fattoriale – Metodo ricorsivo

```
int Fattoriale(int n) {  
    if n=0 then  
        return 1;  
    else  
        return(n * Fattoriale (n-1));  
}
```

Fondamenti di Informatica a.a. 2005/06

Fattoriale

- La versione ricorsiva è più concisa ma non offre alcun vantaggio in termini di velocità di esecuzione (cioè di complessità);
- Dimostriamo la correttezza della seconda versione:
 - *Passo base*: l'algoritmo è certamente corretto per $n=1$, in quanto $1! = 1$;
 - *Passo induttivo*: se l'algoritmo calcola correttamente $(n-1)!$, allora esso calcola correttamente $n!$, che è dato da $n \cdot (n-1)!$

Fondamenti di Informatica a.a. 2005/06

Relazione di ricorrenza

- Calcoliamo la complessità della versione ricorsiva:
 - se $n=0$ $T(n)=1+1=2$;
 - se $n>0$ $T(n)=1+T(n-1)+1=T(n-1)+2$;
- Versione non esplicita di $T(n)$;
- Il metodo più semplice per risolvere questa equazione è espanderla fino ad arrivare ad una espressione in funzione di n e costanti.

Fondamenti di Informatica a.a. 2005/06

Relazione di ricorrenza

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

$$2+(2+T(n-2))=$$

...

$$2+(2+(2+\dots+(2+T(0))))=$$

$$2*n+2 = O(n)$$

- Per relazioni di ricorrenza più complesse, questo metodo è inefficace.

Divide et Impera

- Tecnica di progetto di algoritmi tra le più importanti;
- Consiste nel dividere il problema in sotto problemi di dimensione più piccola, risolvere ricorsivamente tali sottoproblemi, e quindi ottenere dalle soluzioni dei sottoproblemi quella globale.

Divide et Impera

- Fasi:
 1. *Suddivisione del problema*: il problema originale di dimensione n è decomposto in a sottoproblemi di dimensione minore $f_1(n), \dots, f_a(n)$;
 2. *Soluzione ricorsiva degli a sottoproblemi*;
 3. *Combinazione delle soluzioni*.

Divide et Impera

➤ A ciascuna fase sono associati dei costi.
Indichiamo con:

- $Sudd(n)$ tempo del punto 1;
- $Fus(n)$ il tempo del punto 3;
- $T(f_1(n)) + .. + f_a(n)$ il tempo del punto 2.

Divide et Impera

➤ Equazione del tempo di esecuzione:

$$\begin{cases} T(n) = c, & n < n_0 \\ T(n) = T(f_1(n)) + .. + T(f_a(n)) + Sudd(n) + \\ \quad Fus(n) & n \geq n_0 \end{cases}$$

➤ Spesso il tempo di suddivisione è irrilevante o assorbito da altri termini;

Divide et Impera

➤ Indichiamo $d(n) = Sudd(n) + Fus(n)$ come *funzione guida* della equazione di ricorrenza.

$$\begin{cases} T(n) = c, & n < n_0 \\ T(n) = T(f_1(n)) + .. + T(f_a(n)) + d(n) & n \geq n_0 \end{cases}$$

Esempio - Merge sort

- Risolve il problema di ordinamento di un array;
- Fasi:
 - Divide l'array in due metà;
 - Ordina le due metà con due chiamate ricorsive a se stesso;
 - Combina i due semi-array ordinati in un unico array ordinato.

3	-2	0	7	5	4	0	8	-3	-1	9	12	20	5
---	----	---	---	---	---	---	---	----	----	---	----	----	---

Fondamenti di Informatica a.a. 2005/06

Esempio - Merge sort

```
void mergesort(int* a, int p, int r) {
    int q;
    if (p < r) { /* ossia se la lunghezza della porzione
                 maggiore di 1 */
        q = (p+r)/2;
        mergesort(a,p,q); /*ordina la prima porzione */
        mergesort(a,q+1,r); /* ordina la seconda
                             porzione */
        merge(a,p,q,r); /* combina le due porzioni */
    }
}
```

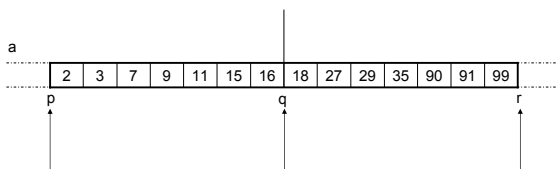
Fondamenti di Informatica a.a. 2005/06

Esempio - Merge sort

- Per fare ciò l'algoritmo si poggia ad una funzione di merge che serve ad ordinare gli elementi di due porzioni adiacenti già ordinate separatamente.
- La funzione merge prende come argomenti:
 - L'array a
 - La posizione di partenza p e l'ultima q della prima porzione ordinata
 - L'ultima posizione r della seconda porzione ordinata (che ha come posizione di partenza q+1)

Fondamenti di Informatica a.a. 2005/06

Esempio - Merge sort



Fondamenti di Informatica a.a. 2005/06

Esempio - Funzione Merge

```
void merge(int* a, int p, int q, int r) {
    int b[MAX], i, j, k; /* MAX denota la dimensione
                        massima che a può avere. */
    i=p; j=q+1; /* i e j "puntano" all'inizio delle due
                porzioni */
    for (k=0; k<(r-p+1); k++) {
        if ((i <= q) && ((j>r) || (a[i]<a[j])))
            b[k]=a[i++];
        else b[k]=a[j++];
    }
    for (k=0; k<(r-p+1); k++)
        a[p+k]=b[k];
}
```

Fondamenti di Informatica a.a. 2005/06

Esempio – Complessità Merge

Assegnamenti esterni	2	+
Test I° for	n+1	+
Numero cicli for	n*(
Corpo for	1+1)+	
Test II° for	n+1	+
Numero cicli for * corpo	n*1	

5n + 4

Fondamenti di Informatica a.a. 2005/06

Esempio – Complessità Merge sort

- Non è influenzata dalla configurazione dell'input;
- Per semplificare i calcoli supponiamo che la dimensione iniziale dell'array sia una potenza di 2, $n=2^p$;
- Scriviamo la relazione di ricorrenza per la $T(n)$:
 $n=1 \quad T(n)=1$;
 $n>1 \quad T(n)=1+1+2T(n/2)+5n+4$.

Fondamenti di Informatica a.a. 2005/06

Esempio – Complessità Merge sort

- $n=1 \quad T(n)=1$
 $n>1 \quad T(n)=2T(n/2)+5n+6$
- Questa espressione della complessità è in funzione di se stessa.
- Proviamo a risolverla per espansioni successive.
- Troviamo **$T(2^q)=2^q+5 \cdot q2^q+6 \cdot (2^q-1)$**

Fondamenti di Informatica a.a. 2005/06

Esempio – Complessità Merge sort

- Essendo $q=\log_2 n$:
 $T(n)=n + 5n \log n + 7n - 6 = 5n \log n + 8n - 6$
- Il mergesort ha complessità **$\Theta(n \log n)$**

Fondamenti di Informatica a.a. 2005/06

Altro esempio di risoluzione

- Data una formula di ricorrenza, supponiamo sia nota la forma generale della funzione $T(n)$ da determinare. Sarà sufficiente trovare i parametri incogniti.

Fondamenti di Informatica a.a. 2005/06

Altro esempio di risoluzione

$$\begin{cases} T(1) = 7 \\ T(n) = 2T(\frac{n}{2}) + 8 \end{cases}$$

- Si ipotizzi che la soluzione abbia la forma: $T(n) = an + b$. Occorre determinare a e b , sapendo che:

$$\begin{cases} T(1) = a + b = 7 \\ T(n) = an + b = 2T(\frac{n}{2}) + 8 = \\ \quad 2(a\frac{n}{2} + b) + 8 = an + 2b + 8 \end{cases}$$

Fondamenti di Informatica a.a. 2005/06

Altro esempio di risoluzione

- Da cui $b = -8$ e $a = 15$
- $T(n) = 15n - 8$

Fondamenti di Informatica a.a. 2005/06

Caso Divide et Impera

- Se indichiamo con a il numero dei sottoproblemi e con n/b la dimensione di ogni sottoproblema, allora il tempo di esecuzione di un algoritmo che segue la tecnica divide et impera ha la seguente equazione di ricorrenza:

$$\begin{cases} T(1) = c \\ T(n) = aT(\frac{n}{b}) + d(n) \end{cases}$$

Con $n=b^k$, $k>0$ e $d(n)$ tempo di divisione e fusione dei sottoproblemi, moltiplicativa.

Fondamenti di Informatica a.a. 2005/06

Teoremi

- Confrontando a con $d(b)$, si dimostra che se:

$$a > d(b) : T(n) = O(n^{\log_b a})$$

$$a < d(b) : T(n) = O(n^{\log_b d(b)})$$

$$a = d(b) : T(n) = O(n^{\log_b d(b)} \log_b n)$$

- Inoltre se $d(n) = n^p$, si ha:

$$a > d(b) : T(n) = O(n^{\log_b a})$$

$$a < d(b) : T(n) = O(n^p)$$

$$a = d(b) : T(n) = O(n^p \log_b n)$$

Fondamenti di Informatica a.a. 2005/06

Esempi

$T(1) = c$	a	b	$d(b)$	
$T_1(n) = 2T(n/2) + n$	2	2	2	$O(n \log n)$
$T_2(n) = 4T(n/2) + n$	4	2	2	$O(n^2)$
$T_3(n) = 4T(n/2) + n^2$	4	2	4	$O(n^2 \log n)$
$T_4(n) = 4T(n/2) + n^3$	4	2	8	$O(n^3)$

Fondamenti di Informatica a.a. 2005/06
