

Costi di esecuzione

Luca Tagliavini

February 25 - March 1, 2021

Contents

0.1 Def: Consto di esecuzione

Un algoritmo \mathcal{A} ha *consto di esecuzione* $O(f(n))$ su istanze di ingresso di dimensione n rispetto a una certa *risorsa di calcolo* se la quantità $r(n)$ di risorsa sufficiente per eseguire \mathcal{A} su una qualunque istanza di dimensione n verificata dalla relazione $r(n) = O(f(n))$.

Per noi *risorsa di calcolo* significa **tempo di esecuzione** o **occupazione di memoria**.

0.2 Def: Complessita'

Un problema \mathcal{P} ha una *complessita'* $O(f(n))$ rispetto a una data risorsa di calcolo se esiste un algoritmo che risolve \mathcal{P} il cui costo di esecuzione rispetto a quella risorsa è $O(f(n))$.

0.3 Selezione del caso peggiore

Prendendo ad esempio un codice formato da operazioni *non elementari*, come if condizionali o cicli for/while, dobbiamo sempre metterci nel caso di considerare l'input peggiore. Ad esempio in quest codice:

```
algoritmo k() {  
    if cond then  
        caso_true  
    else  
        caso_false  
}
```

Avremo:

- $\text{cond} = O(f(n))$
- $\text{caso_true} = O(g(n))$
- $\text{caso_false} = O(h(n))$

E sceglieremo dunque come costo computazionale dell'algoritmo k :

$$O(\max \{f(n), g(n), h(n)\})$$

0.4 Analisi per casi

Sia \mathcal{I}_n l'insieme di tutte le possibili istanze di input di lunghezza n . Sia $T(I)$ il tempo di esecuzione dell'algoritmo sull'istanza $I \in \mathcal{I}_n$

- **worst case:** $T_{\text{worst}}(n) = \max_{I \in \mathcal{I}_n} T(I)$
- **best case:** $T_{\text{best}}(n) = \min_{I \in \mathcal{I}_n} T(I)$
- **average case:** $T_{\text{avg}}(n) = \sum_{I \in \mathcal{I}_n} T(I)P(I)$
dove $P(I)$ è un peso in percentuale che varia in base alla probabilità che un determinato input venga dato in pasto all'algoritmo.

0.5 Analisi algoritmi ricorsivi (Teorema dell'esperto)

$$T(n) = \begin{cases} c_1 & \text{se } n = 0 \\ a \cdot T(n/b) + c_2 \cdot n^\beta & \text{se } n = 0, c_2 > 0 \end{cases}$$

Siano

- c_1 il *costo del caso base* e c_2 il costo del caso ricorsivo
- a il *numero di chiamate ricorsive*
- b il *numero di partizioni dell'input*
ovvero, grandezza dei dati passati alle sottochiamate ricorsive
- $\alpha = \frac{\log_2 a}{\log_2 b}$
- β e' l'esponente della complessita' aggiuntiva per ogni sottochiamata ricorsiva. $\beta = 0$ se la complessita' e' costante.

A questo punto confronto i valori di α e β per vedere il caso in cui sono:

1. $T(n) = \Theta(n^\alpha)$ se $\alpha > \beta$
2. $T(n) = \Theta(n^\alpha \log_2 n)$ se $\alpha = \beta$
3. $T(n) = \Theta(n^\beta)$ se $\alpha < \beta$

ATTENZIONE: il teorema non e' applicabile ad algoritmi ricorsivi che non effettuano *partizioni bilanciate* (ossia quando avendo due o piu' sottochiamate ricorsive effettuano chiamate con n della stessa grandezza). Ad esempio fibonacci che chiama $fib(n-1)$ e $fib(n-2)$ *non effettua partizioni bilanciate*.

0.6 Esempio di algoritmo ricorsivo

Analizziamo l'algoritmo di ricerca binaria tramite il *master theorem*:

Abbiamo $T(n) = T(n/2) + O(1)$. Da cui $a = 1, b = 2$.

Vogliamo che n^β sia costante, dunque $\beta = 0$.

Dunque $\alpha = \frac{\log_2 1}{\log_2 2} = 0$. Siamo nel caso $\alpha = \beta$:

Ne segue $T(n) = \Theta(n^0 \cdot \log_2 n) = \Theta(n \log_2 n)$.