Teoria della complessità

Introduzione alla complessità dei problemi

Donato Malerba

Complessità di un problema

Finora abbiamo discusso della complessità degli *algoritmi*, cioè di procedimenti risolutivi di problemi. Tuttavia, ha senso parlare anche di complessità intrinseca dei *problemi*. Questa viene definita come segue:

Def. di complessità di un problema

Un problema ha complessità di ordine $g(n)$ se qualunque algoritmo risolutivo del problema ha una complessità di ordine $\Omega(g(n))$.

In pratica, se un problema $\mathcal{P}$ ha una sua complessità $g(n)$, non si potrà mai trovare un algoritmo per $\mathcal{P}$ che abbia una complessità inferiore a $g(n)$.

Complessità di un problema

Esempio: Il problema della ricerca del massimo in un array di dimensione n ha una complessità temporale intrinseca pari a n. Quindi è inutile cercare gli algoritmi che risolvono il problema in tempi proporzionali a $\log n$ o addirittura in tempi costanti.

Ma come determinare la complessità di un problema $\mathcal{P}$?

Per dimostrare che $\mathcal{P}$ ha complessità pari a $g(n)$ non basta trovare un algoritmo con complessità $\Theta(g(n))$. Questo, infatti, ci garantisce che il problema ha una soluzione con complessità *almeno* g(n), ma nulla impedisce che si possa trovare in futuro un altro algoritmo di complessità inferiore a g(n).

Complessità di un problema

In generale, ogni algoritmo stabilisce un limite *superiore* per la complessità del particolare problema risolto. La determinazione di un limite *inferiore* per la complessità di un problema richiede invece la dimostrazione che *tutti* gli algoritmi risolutivi di quel problema abbiano una complessità peggiore del limite inferiore stabilito.

Esempio: Se un problema $\mathcal{P}$ fosse risolvibile mediante solo tre algoritmi, A1, A2 e A3 di complessità:

A1: $f1(n) = n^2 \log_2 n$

A2: $f2(n) = n \log_2 n$

A3: $f3(n) = a^n$ con $a>1$

allora $\mathcal{P}$ sarebbe un problema di complessità $n \log_2 n$.

Complessità di un problema

Nella pratica non possiamo pensare di valutare la complessità di un problema $\mathcal{P}$ cercando tutti gli algoritmi che risolvono $\mathcal{P}$, perché ce ne potrebbero essere infiniti.

Si deve invece ricorrere ad una vera e propria *dimostrazione* matematica, se si vogliono stabilire dei limiti *significativi* per la complessità dei problemi.

Ad esempio, si può ragionevolmente affermare che il problema di ordinare un array di n dati ha complessità pari almeno ad n, in quanto ogni algoritmo di ordinamento deve necessariamente esaminare almeno una volta ciascuno degli elementi dell'array.

Complessità di un problema

Appare comunque poco realistico pensare che si possa effettuare un ordinamento con complessità lineare rispetto ad n.

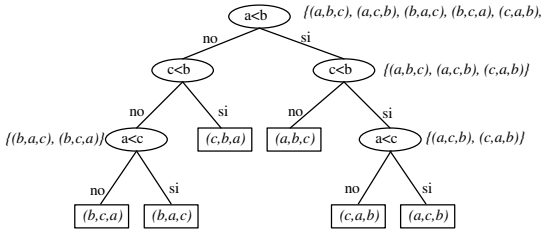
In effetti gli algoritmi di ordinamento che abbiamo esaminato avevano almeno complessità in tempo di ordine $O(n\log n)$ nel caso pessimo.

In casi simili si dice che il limite stabilito è *poco significativo*.

Di seguito, si dimostra che un limite significativo per il problema dell'ordinamento di n dati è $n\log n$.

Problema dell'ordinamento

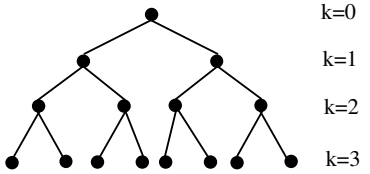
Consideriamo il problema di ordinare tre elementi *a*, *b* e *c*. Ognuna delle 6 permutazioni di questi elementi potrebbe rappresentare l'ordinamento corretto. Le possibilità si possono via scartare per confronti successivi fino ad arrivare alla soluzione, come mostrato nel seguente *albero decisionale*.



Ogni nodo nell'albero corrisponde ad un confronto e ad esso è associato l'insieme delle permutazioni ancora accettabili prima di effettuare tale confronto. Ogni confronto ha due possibili esiti, ognuno dei quali consente di scartare alcune possibilità. Il procedimento continua finché rimane una sola possibilità, che è la soluzione.

Problema dell'ordinamento

Analizziamo adesso il problema generale di ordinare *n* dati. Ad ogni livello dell'albero, il numero dei nodi generalmente raddoppia, quindi al *k*-esimo livello si hanno al più 2^k nodi. Ma se un albero ha *k* livelli, allora vuol dire che per arrivare ad una soluzione occorre effettuare *k* confronti.



D'altronde le possibili soluzioni sono *n*!, ovvero tutte le permutazioni degli elementi di partenza. Quindi per avere la certezza di arrivare alla soluzione in ogni caso, l'albero deve avere *n*! nodi soluzione, e ciò è possibile se il numero *k* di confronti soddisfa la seguente disuguaglianza:

$$n! \leq 2^k$$

Problema dell'ordinamento

Poiché la funzione logaritmo è strettamente crescente, prendendo il logaritmo di ambo i membri la disuguaglianza si conserva, dunque si ha che:

$$\log(n!) \leq \log(2^k) = k$$

ma per l'approssimazione di *Stirling*:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

possiamo scrivere:

$$k \geq \log_2(n!) \approx n\log_2 n - n\log_2 e + \frac{1}{2}\log_2 n + \frac{1}{2}\log_2 2\pi$$

Quindi un limite inferiore al numero di confronti è dato da:

$$g(n) = n\log_2 n - n\log_2 e + \frac{1}{2}\log_2 n + \frac{1}{2}\log_2 2\pi$$

Possiamo concludere che il problema di ordinare n dati ha una complessità $n\log_2 n$.

Algoritmo ottimo in ordine di grandezza

- Un algoritmo che ha la stessa complessità intrinseca di un problema si dice *ottimo in ordine di grandezza*.
- L'algoritmo di ordinamento *Mergesort* è un esempio di algoritmo ottimo in ordine di grandezza.