**COMPLESSITÀ COMPUTAZIONALE**

Complessità di un algoritmo 🡪 misura delle risorse di calcolo consumate durante la computazione;

Teoria della complessità computazionale 🡪 Branca dell’informatica che studia l’uso delle risorse di calcolo necessarie per la computazione di algoritmi e per la soluzione di problemi.

Un algoritmo può essere valutato in base a misure statiche (linee di codice, numero di cicli) o in base a misure dinamiche (tempo di computazione, utilizzo di memoria. Il tempo è più importante dello spazio perché lo spazio può essere riutilizzato, il tempo perso è perso per sempre ;-) ).

COSI DI SERVE PER DEFINIRE LA COMPLESSITÀ ?

* Un modello di costo;
* I parametri in funzione dei quali deve essere definita la complessità.

Per misurare la complessità di una algoritmo, dobbiamo avere un modello di costo che ci permetta di valutare istruzioni e comandi, assegnando un costo temporale ad ognuno di essi, per poter ottenere un risultato approssimato, che ci indichi quanto l’algoritmo costa (quanto è efficiente).

MODELLO DI COSTO IN TEMPO

Criterio di costo logaritmico 🡪 In un calcolatore una cella di memoria ha ovviamente capacità limitata. Si assume che per contenere numeri arbitrariamente grandi, abbiamo bisogno di un numero di celle di memoria proporzionale al logaritmo dei numeri stessi. Per eseguire una istruzione si deve pagare un costo proporzionale al logaritmo degli operandi.

Criterio di costo uniforme 🡪 è un criterio più appropriato rispetto a quello logaritmico se ogni numero che incontriamo è memorizzabile in una parola.

MODELLO DI COSTO IN SPAZIO

Se immaginassimo di adottare un modello di costo in termini di spazio, la sommatoria di 10 numeri avrebbe complessità 12 (1 scrittura per la inizializzazione di somma, una scrittura per l’inizializzazione dell’indice di scansione, 10 scritture in memoria per sommare i 10 numeri). In questo caso si sta facendo una assunzione di costo uniforme e il risultato sarà ovviamente approssimato.

Quasi sempre il tempo di computazione di un algoritmo dipende dalla configurazione dei dati di ingresso (cosa significa questa cosa? Sembra abbastanza banale osservare che l’algoritmo che calcola il fattoria di un numero sarà meno costoso in termini di tempo sa calcola il fattoriale di 3 piuttosto che quello di 1000). Una buona modalità per approcciarsi alla valutazione della dimensione dell’input potrebbe essere quella di prescindere dal tipo di input in ingresso e di soffermarsi sulla quantità di memoria utilizzata per rappresentare appunto i dati in ingresso. Questo potrebbe far apparire lineare il costo di un algoritmo piuttosto che esponenziale.

Funzione di complessità 🡪 Determinare la complessità di un algoritmo significa determinare una funzione di complessità f(n) che fornisca la misura di tempo (o di spazio) al variare della dimensione dell’input n.

Operazione dominante 🡪 Una operazione dominante è quando dato t(n) che indica il numero di volte che essa si ripete in seguito all’input n, per una certa costante c si verifica che c\*t(n) >= f(n)

Una volta che abbiamo la nostra funzione di complessità essa va studiata asintoticamente. Lo studio asintotico è il fulcro della teoria della complessità perché ci permette di confrontare due funzioni di complessità su uno stesso algoritmo e stabilire quale delle due è la più efficiente. Vengono tralasciate le costanti (in termini informatici, potremmo dire che se abbiamo 4n invece che n, il 4 rappresenta un hardware quattro volte più lento rispetto ad un altro che sta eseguendo il programma in cui è contenuto l’algoritmo, e quindi ci mette 4 volte il tempo che ci metterebbe un altro hardware).

Possiamo identificare 4 ordini di grandezza.

**ORDINI DI GRANDEZZA**

ORDINE O 🡪 Una funzione f(n) è di ordine O(g(n)) se g(n) indica un limite superiore al comportamento asintotico di f(n), cioè che f(n) non si comporta peggio di g(n), al massimo si comporta male quanto g(n). Per ogni n maggiore di un certo n0, f(n) <= cg(n).

g(n) è maggiorante di f(n).

Il rapporto fra f(n) e g(n) o fa zero (perché g, che rappresenta il denominatore, cresce più velocemente) o da come risultato una costante se si comportano allo stesso modo.

(Dimostrare che un polinomio di grado m appartiene sempre all’ordine O(n^m)).

ORDINE o 🡪 Una funzione f(n) è di ordine o(g(n)) se g(n) indica un limite strettamente superiore al comportamento asintotico di (fn), cioè che f(n) si comporta sempre meglio di g(n). Il loro rapporto darà come risultato sempre e solo zero (constatabile in maniera ovvia, dato che g(n) al denominatore sarà sempre di un ordine superiore a f(n)).

ORDINE Ω **🡪** Se g(n) indica un limite inferiore di f(n), cioè al massimo f si comporta bene quanto g da un certo n0 in poi. Il limite del rapporto o vale infinito o vale una costante > 0, perché la funzione f diverge più velocemente

ORINDE ω 🡪 Se è una minorante stretta di f. Il limite ad infinito del rapporto sarà sempre infinito

ORDINE Θ 🡪 Se f è sia di ordine O(g(n)) e sia di ordine Ω(g(n)) allo stesso tempo. Cioè il limite ad infinito del rapporto è sempre una costante, e praticamente significa che da un certo n0 in f(n) si comporta esattamente come g(n).

**RISOLVERE RELAZIONI DI RICORRENZA**

Costo delle chiamate ricorsive + lavoro di combinazione

In base al lavoro di combinazione possiamo definire delle vere e proprie classi di relazioni di ricorrenza in base alle caratteristiche del lavoro di combinazione svolto.

**TEORIA DELLA COMPUTAZIONE SEQUENZIALE**

Nonostante le varie suddivisioni in svariate classi di complessità computazionale, le due macro suddivisioni che si possono fare in base alle complessità di un algoritmo sono quelli appartenenti a ordine di complessità POLINOMIALE e quelli appartenenti ad un ordine di complessità ESPONENZIALE

POLINOMIALE 🡪 O(n^k)

ESPONENZIALE 🡪OMEGA(a^g(n)) con a > 1 e g(n) una funzione super logaritmica per n crescente all’infinito.

La teoria della computazione sequenziale afferma che calcolatori diversi possono simularsi a vicenda (in base a quanto afferma la tesi di Church, cioè che due calcolatori sono in grado di risolvere gli stessi problemi) senza però che un algoritmo cambia classe (da polinomiale ad esponenziale e viceversa) per effetto della simulazione. Rappresenta un passo avanti rispetto alla tesi di Church, è più forte, perché due calcolatori possono risolvere lo stesso problema, ed aggiunge anche che gli algoritmi non cambieranno classe.

**COMPLESSITA COMPUTAZIONALE DEI PROBLEMI**

**Schemino:** g(n), come si determina la complessità, esempio ordinamento

Un problema ha complessità di ordine g(n) se l’algoritmo utilizzato per risolvere il problema è di ordine OMEGA(g(n)). Se quindi un problema è di ordine g(n) non si potrà mai trovare un algoritmo con complessità inferiore a g(n) per risolvere tale problema.

Ma come si determina la complessità di un problema? È relativamente semplice stabilire un limite superiore di complessità per la risoluzione di un problema, ma stabilire il limite inferiore è più complicato, perché potrebbero esserci infiniti algoritmi che risolvono il problema con una complessità a noi sconosciuta, che potrebbe essere sempre la più efficiente. Non lo sappiamo. Bisogna quindi ricorrere a vere e proprie dimostrazioni matematiche, per provare che un problema ha una certa g(n) come limite inferiore. Ad esempio per il caso della ricerca del minimo la complessità per risolvere il problema è come minimo una funzione lineare, perché tutti gli n elementi devono essere scanditi nell’array almeno una volta.

Il limite inferiore che troviamo per la risoluzione di un certo problema deve essere comunque un limite inferiore significativo. Supponiamo di dover ordinare un array. Sarebbe difficile immaginare che un array venga ordinato tramite n confronti, cioè scandendo gli elementi una volta sola. È vero che OMEGA(n) costituirebbe un limite inferiore per la complessità del problema, ma sarebbe evidentemente un limite inferiore poco significativo. Nella migliore delle ipotesi un algoritmo di ordinamento aveva complessità n\*logn, questo rappresenta infatti un limite inferiore significativo per i problemi di ordinamento.