Complessita’ Computazionale

1. Introduzione all’efficienza degli algoritmi

La **teoria della complessità computazionale** studia le risorse minime necessarie (principalmente tempo di calcolo e memoria) per la risoluzione di un problema. Valutare l’efficienza di un algoritmo e’ essenziale per determinare i limiti di applicabilita’ in problemi reali. Che senso avrebbe realizzare un programma che impiegherebbe migliaia di anni per restituire una risposta?

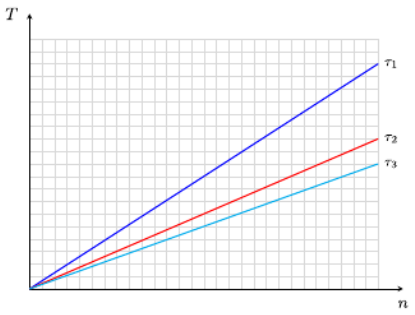
Distinguiamo:

- **Complessità spaziale**: utilizzo delle risorse da parte di un programma.
- **Complessità temporale**: tempo di esecuzione di un programma, misura l’efficienza di esecuzione;

Con “efficienza” infatti non intendiamo solamente il tempo necessario all’esecuzione ma anche l’utilizzo delle altre risorse.

Il tempo di esecuzione di un algoritmo su un calcolatore e’ influenzato da svariati fattori: il linguaggio, il compilatore, l’hardware della macchina, la dimensione della struttura dati su cui l’algoritmo opera... E’ pertanto necessario utilizzare un modello astratto che non dipenda da questi fattori: il modello **RAM (Random Access Machine).**

Uno stesso algoritmo potrebbe infatti impiegare un tempo differente in base alla macchina sulla quale viene eseguito: consideriamo un algoritmo A codificato in HLL che effettui la somma dei primi n numeri naturali... compilato ed eseguito su macchine differenti.



I tempi di esecuzione, al variare di n sono indicati con: $\tau_1(n), \tau_2(n), \tau_3(n)$... e sono sensibilmente diversi tra loro.

Tuttavia presentano un **andamento comune**: sono tutti **lineari con** n, ovvero rispondono alla legge:

$$\tau(n) = c_1 n + c2$$

Si verifica che, per elevati valori di n, l’**andamento asintotico** di τ_n e’ uguale in quanto sempre linearmente proporzionale ad n. Per la caratterizzazione dell’efficienza degli algoritmi si considera pertanto l’andamento asintotico.

2. Modello RAM

- sistema monoprocessore con esecuzione sequenziale, nessun parallelismo
- le istruzioni semplici del linguaggio (operazioni di assegnazioni, aritmetiche, relazionali o logiche) richiedono un tempo unitario di esecuzione, $T(1)$.

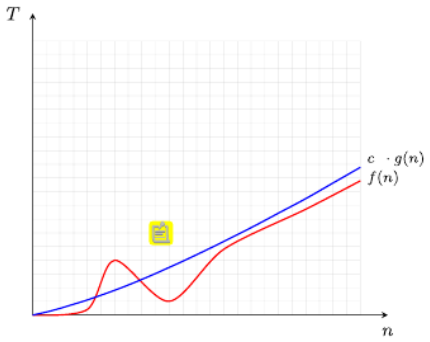
Usando il **modello RAM** si va a **stimare** il tempo di esecuzione $T(n)$ di un determinato algoritmo.

Si definisce **complessita’ computazionale l’ordine di grandezza della funzione** $T(n)$ che rappresenta il numero di istruzioni da eseguire nel modello RAM.

Definire la funzione $T(n)$ e’ essenziale in quanto gli strumenti dell’analisi asintotica possono essere applicati solamente a delle funzioni. Non sarebbe stato possibile quantificare direttamente in termini temporali.

3. Notazioni Asintotiche

3.1 Notazione $O(g(n))$

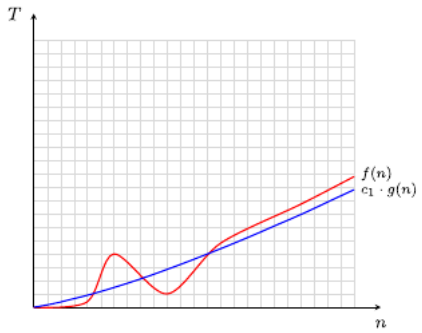


$f(n) \in O(g(n)) \iff \exists c, n_0 > 0 : \forall n > n_0, 0 \leq f(n) \leq$

$f(n)$ appartiene ad $O(g(n))$ se, a partire da un certo valore n_0, $g(n) * c$ **maggiora** la funzione $f(n)$ per tutti i valori successivi di n.

$g(n)$ rappresenta un **limite (lasco) superiore** per $f(n)$.

3.2 Notazione $\Omega(g(n))$

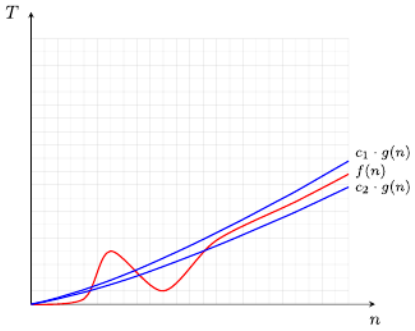


$f(n) \in \Omega(g(n)) \iff \exists c, n_0 > 0 : \forall n > n_0, 0 \leq c * g(n$

$f(n)$ appartiene ad $\Omega(g(n))$ se, a partire da un certo valore n_0, $g(n) * c$ **minora** la funzione $f(n)$ per tutti i valori successivi di n.

$g(n)$ rappresenta un **limite (lasco) inferiore** per $f(n)$.

3.3 Notazione $\Theta(g(n))$



$f(n) \in \Theta(g(n)) \iff \exists c_1, c_2, n_0 > 0 : \forall n > n_0,\ c_1 * g($

A partire da un certo valore n_0, $f(n)$ e' **compresa** tra $c_1 g(n)$ e $c_2 g(n)$. Cio' significa che al crescere di n, $f(n)$ e $g(n)$ crescono allo stesso modo.

Costituisce un **limite stretto.**

3.3.1 Best, Worst, Average case

Il tempo di esecuzione, oltre che della dimensione della struttura dati, potrebbe dipendere anche dallo specifico **valore in ingresso**. In un algoritmo di ordinamento di un vettore, il tempo e' strettamente dipendente dalla condizione in ingresso del vettore, ovvero il caso in cui il vettore in ingresso sia o meno gia' ordinato.

```
if (condizione)
   a=b;
else {
   somma = 0;
   for (i=0 ; i<n ; i++)
      somma = somma + 1;
}
```

E' possibile distinguire 3 casi differenti (data una struttura dati d di dimensione n):

- **Best Case**: corrisponde alle configurazioni di d che danno luogo al **tempo minimo**;
- **Worst Case:** configurazioni di d che danno luogo al **tempo massimo;**
- **Average Case**: valori che si ottengono in configurazioni normali, ne' migliori ne' peggiori;

3.4 Proprieta' delle notazioni

La complessita' di un algoritmo si determina a partire dai blocchi che costituiscono lo stesso, valutando singolarmente la complessita' di tutte le strutture di controllo di cui si compone. E' pertanto molto utile introdurre l'operazione di somma.

$$T(n) = \Theta(f_1(n)) + \Theta(f_2(n)) = \Theta(max(f_1(n), f_2(n)))$$
$$T(n) = O(f_1(n)) + O(f_2(n) = O(max(f_1(n), f_2(n)))$$
$$T(n) = \Omega(f_1(n)) + \Omega(f_2(n)) = \Omega(min(f_1(n), f_2(n)))$$

ESEMPIO:

$$T(n) = O(n^2) + O(n^3) = O(max(n^2, n^3)) = O(n^3)$$

Se una funzione $f(n)$ appartiene a $\Theta(g(n))$, allora appartiente sia a $O(g(n))$ che a $\Omega(g(n))$. Vale il viceversa.

$$f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \text{ e } f(n) \in \Omega(g(n))$$

4. Complessita' dei costrutti

4.1 Costrutto sequenziale

Nel modello RAM le istruzioni semplici (operazioni di assegnamento, logici, relazionali, aritmetici) offerte dal linguaggio hanno complessitá costante $\Theta(1)$.

4.2 Costrutto selettivo

```
if (condizione)
   { <istruzioni - then > }
else
   { <istruzioni - else >
}
```

Si considera:

- T_{cond} : tempo per la verifica della condizione
- T_{then} : tempo per l'esecuzione del ramo then
- T_{else} : tempo per l'esecuzione del ramo else

$$T_{best} = \Theta\ (min(T_{cond} + T_{then}\ ;\ T_{cond} + T_{else}) = \Omega\ (min(T_{cond} + T_{then}\ ;\ T_{cond} + T_{else})$$

$$T_{worst} = \Theta\ (max(T_{cond} + T_{then}\ ;\ T_{cond} + T_{else}) = O\ (max(T_{cond} + T_{then}\ ;\ T_{cond} + T_{else})$$

4.3 Costrutto iterativo

4.3.1 Ciclo for

```
for (init ; cond ; inc){
   <istruzioni - for>
}
```

Si considera:

- T_{init}
- T_{cond}
- T_{inc}
- T_{corpo}

$$T_{best} = \Omega(T_{init} + k(T_{corpo,best} + T_{cond} + T_{inc}))$$

$$T_{worst} = O(T_{init} + k(T_{corpo,worst} + T_{cond} + T_{inc}))$$

Nella maggior parte dei casi $T_{init}, T_{cond}, T_{inc}$ sono costanti pertanto a fare la differenza sono le istruzioni presenti nel corpo del for T_{corpo}. Conoscendo i **limiti stretti** ed assumendo $T_{init}, T_{cond}, T_{inc}$ unitari:

$$T_{best} = \Theta(kT_{corpo,best})$$
$$T_{worst} = \Theta(kT_{corpo,worst})$$

4.3.1 Ciclo while

```
while ( cond )
   < istruzioni - ciclo >
```

Si considera:

- T_{corpo}
- T_{cond}

$$T_{best} = \Omega(k_{min}(T_{corpo,best} + T_{cond}))$$
$$T_{worst} = O(k_{max}(T_{corpo,worst} + T_{cond}))$$

Conoscendo i limiti stretti e assumento che T_{cond} sia unitario:

$$T_{best} = \Theta(k_{min}T_{corpo,best})$$
$$T_{worst} = \Theta(k_{max}T_{corpo,worst})$$

4.4 Chiamate a funzione

Il tempo e' dato dalla somma di 2 contributi:

- uno derivante dalla **chiamata a funzione** (non sempre il tempo e' UNITARIO! E' strettamente dipendente dalla dimensione dei parametri);
- Il secondo dovuto all'esecuzione della funzione ovvero al **corpo della stessa**.

4.4.1 Funzioni Ricorsive

Si esprime il tempo incognito $T(n)$ dell'esecuzione di R come somma di due contributi:

- tempo $f(n)$: istruzioni che non contengono chiamate ricorsive;
- tempo $T(k)$: chiamata ricorsiva

Da cui deriva la **relazione di ricorrenza**:

$$T(n) = \begin{cases} c_1 & \text{per } n = 1 \\ D(n) + C(n) + \sum_{i=0}^{k} T(k_i) & \text{per } n > 1, k_i < n \end{cases}$$

- $D(n)$: tempo necessario alla fase di "Divide";
- $C(n)$: tempo necessario alla fase di "Combina";
- $\sum_{i=0}^{k} T(k_i)$: tempo necessario ad eseguire le istanze ricorsive sulla struttura dati k_i.

4.4.1.1 Ricorrenze notevoli

Divisione della struttura in due parti uguali con chiamata ricorsiva su una **sola parte** e tempo di combinazione e divisione **costante**:

$$T(n) = \begin{cases} c_1 & \text{per } n = 1 \\ T(\frac{n}{2}) + c_2 & \text{per } n > 1 \end{cases}$$

Divisione della struttura dati in due parti eguali, con chiamata ricorsiva su **entrambe le parti** e tempo di combinazione e divisione **costante**:

$$T(n) = \begin{cases} c_1 & \text{per } n = 1 \\ 2T(\frac{n}{2}) + c_2 & \text{per } n > 1 \end{cases}$$

$$T(n) \in O(\log n)$$

Divisione della struttura dati in due parti eguali, con invocazione ricorsiva su
una
sola delle parti e tempo di combinazione e divisione **lineare**:

$$T(n) = \begin{cases} c_1 & \text{per } n = 1 \\ T(\frac{n}{2}) + nc_2 & \text{per } n > 1 \end{cases}$$

$$T(n) \in O(n)$$

$$T(n) \in O(n)$$

Divisione della struttura dati in due parti eguali, con invocazione ricorsiva su

entrambe le parti e tempo di combinazione e divisione **lineare**:

$$T(n) = \begin{cases} c_1 & \text{per } n = 1 \\ 2T(\frac{n}{2}) + nc_2 & \text{per } n > 1 \end{cases}$$

$$T(n) \in O(n \log_2 n)$$

Divisione della struttura dati in due parti di dimensioni 1 ed $n - 1$, con un'unica chiamata ricorsiva e tempo di combinazione e divisione **costante**:

$$T(n) = \begin{cases} c_1 & \text{per } n = 1 \\ T(n-1) + c_2 & \text{per } n > 1 \end{cases}$$

$$T(n) = O(n)$$

La risoluzione delle ricorrenze notevoli viene effettuata applicando il **metodo iterativo** ossia si cerca il valore per il quale la ricorrenza si chiude affinando per sostituzioni successive.