Progettazione di Algoritmi **Anno Accademico 2021–2022**

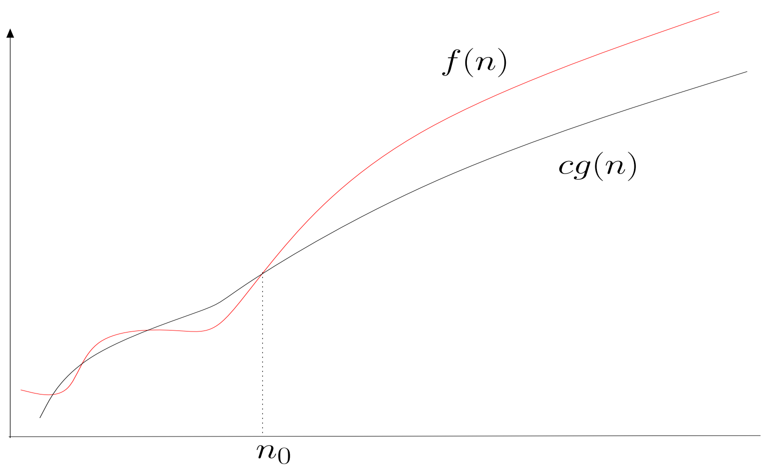
Lezione 4

Ugo Vaccaro

Iniziamo con l'introdurre la notazione Ω, che ci sarà utile quando vorremo valutare limitazioni *inferiori* al tempo di esecuzione di algoritmi (useremo invece la notazione O quando vorremo valutare limitazioni *superiori* al tempo di esecuzione di algoritmi).

Date funzioni $f : n \in \mathbb{N} \to f(n) \in \mathbb{R}_+$, $g : n \in \mathbb{N} \to g(n) \in \mathbb{R}_+$, diremo che $f(n) = \Omega(g(n))$ se e solo se esistono costanti $c > 0$ e $n_0 \in \mathbb{N}$ tale che $f(n) \geq cg(n)$, per ogni $n \geq n_0$.

Informalmente, $f(n) = \Omega(g(n))$ se la funzione $f(n)$ cresce tanto velocemente almeno quanto cresce la funzione $g(n)$. Graficamente, abbiamo una situazione siffatta, ovvero il grafico della funzione $f(n)$, dal punto n_0 in poi, si trova sempre sopra il grafico della funzione $cg(n)$, per qualche costante $c > 0$



Vediamo un esempio. Sia $f(n) = n^2 - 2n$, $g(n) = n^2$ e vogliamo provare che $n^2 - 2n = \Omega(n^2)$. In accordo alla definizione, occorrerà provare che esiste una costante $c > 0$ ed un numero intero n_0 tale che $n^2 - 2n \geq cn^2$, per ogni $n \geq n_0$. Osserviamo che

$$n^2 - 2n \geq cn^2 \Leftrightarrow n^2 - cn^2 \geq 2n \Leftrightarrow n^2(1-c) \geq 2n \Leftrightarrow n(1-c) \geq 2 \Leftrightarrow n \geq 2/(1-c),$$

per cui basterà scegliere c come un qualsiasi valore < 1. Ad esempio, potremmo scegliere $c = 1/2$ ed avremmo quindi che $n^2 - 2n \geq cn^2$ per ogni valore di $n \geq 4$.

Osserviamo ora che valgono le seguenti relazioni

$$\begin{aligned} f(n) = \Omega(g(n)) &\Leftrightarrow \exists c, n_0 : f(n) \geq cg(n),\ \forall n \geq n_0 \\ &\Leftrightarrow \exists c, n_0 : g(n) \leq \frac{1}{c} f(n),\ \forall n \geq n_0 \\ &\Leftrightarrow g(n) = O(f(n)). \end{aligned}$$

Pertanto, per provare che $f(n) = \Omega(g(n))$ basterà provare che $g(n) = O(f(n))$. Sulla base, quindi, di quanto già provato per la notazione asintotica O, possiamo dire che

$$\sqrt{n} = \Omega(\log n)),\ n = \Omega(\sqrt{n}),\ n^{k+1} = \Omega(n^k),\ 2^n = \Omega(n^k),\ n! = \Omega(2^n),\ n^n = \Omega(2^n).$$

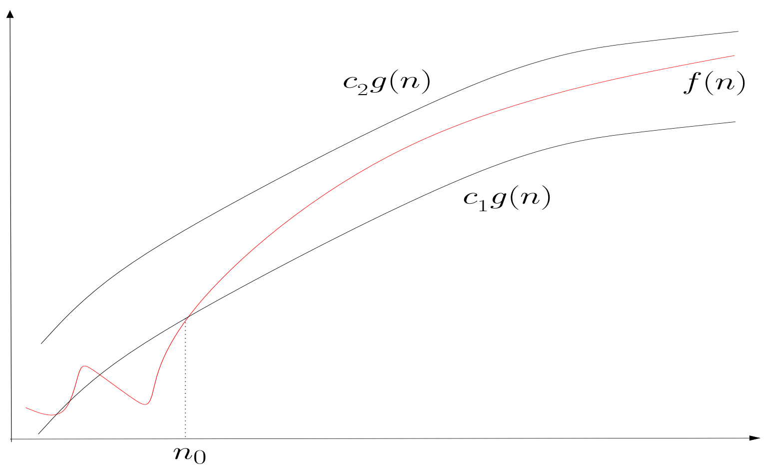
Ad esempio, se volessimo provare che $n^2 - \sqrt{n}\log n = \Omega(n^2)$, dovremmo provare che $\exists c, n_0$ tale che $n^2 - \sqrt{n}\log n \geq cn^2$, $\forall n \geq n_0$. A tal fine, osserviamo che

$$n^2 - \sqrt{n}\log n \geq n^2 - \sqrt{n}\sqrt{n} = n^2 - n \geq n^2 - \frac{1}{2}n^2 = \frac{1}{2}n^2,\ \forall n \geq 2.$$

Useremo infine la notazione asintotica Θ se $f(n)$ e $g(n)$ crescono alla stessa velocità. Più precisamente

$$\begin{aligned} f(n) = \Theta(g(n)) &\Leftrightarrow f(n) = O(g(n)) \& f(n) = \Omega(g(n)) \\ &\Leftrightarrow \exists c_1, c_2, n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall n \geq n_0. \end{aligned}$$

Graficamente



Vediamo un esempio. Cerchiamo di provare che $4n^2 + \log n = \Theta(n^2)$. Osserviamo innanzitutto che $4n^2 + \log n = \Omega(n^2)$ in quanto $4n^2 + \log n \geq n^2$. Inoltre, $4n^2 + \log n = O(n^2)$ in quanto $4n^2 + \log n \leq 4n^2 + n \leq 5n^2$.

Utilizzeremo la notazione asintotica Θ quando saremo in grado di dare una valutazione precisa della velocità di crescita di una funzione. Quindi, sarà sicuramente corretto dire che, ad esempio, $n^2 + n = O(n^2)$, ma sarà preferibile dire (in quanto è una affermazione più precisa) che $n^2 + n = \Theta(n^2)$, in quanto sappiamo che vale sia $n^2 + n = O(n^2)$ che $n^2 + n = \Omega(n^2)$.

Vediamo qualche esercizio.

1. Sia $g(n) = n + 2n^3 - 3n^4 + 4n^5$. Dire quali delle seguenti affermazioni sono vere e quali sono false.
   (a) $g(n) = \Omega(n \log n)$ `Vero`
   (b) $g(n) = \Theta(5n^6)$ `Falso`
   (c) $g(n) = O(n^{10})$ `Vero`
   (d) $g(n) = \Omega(n^5)$ `Vero`
2. Sia $g(n) = n \log n + 2n^3 - 3n^2$. Dire quali delle seguenti affermazioni sono vere e quali sono false.

(a) $g(n) = O(n \log n)$ `Falso`

(b) $g(n) = O(n^3)$ `Vero`

(c) $g(n) = O(n^2)$ `Falso`

(d) $g(n) = O(n^4)$ `Vero`

3. Sia $f(n) = 4n^2 + n + 3$. Dire quali delle seguenti affermazioni sono vere e quali sono false.

   (a) $f(n) = O(n^2)$ `Vero`

   (b) $f(n) = O(3n^2 + n + 3)$ `Vero`

   (c) $f(n) = \Omega(5n^2 + n + 3)$ `Vero`

   (d) $f(n) = \Omega(n^2)$ `Vero`

4. (a) Trovare una funzione $g(n)$ tale che la funzione $f(n) = n^3 \log n^4 + 2n^4 + 80$ sia $f(n) = O(g(n))$. Si *dimostri* che la funzione scelta soddisfi il requisito.

   (b) Trovare una funzione $g(n)$ tale che la funzione $f(n) = n + 3n^2 + 4n^3$ sia $f(n) = \Omega(g(n))$. Si *dimostri* che la funzione scelta soddisfi il requisito.

5. (a) Trovare una funzione $g(n)$ tale che la funzione $f(n) = n^3 \log n^4 + 2n^4 + 80$ sia $f(n) = O(g(n))$. Si *dimostri* che la funzione scelta soddisfi il requisito.

   (b) Trovare una funzione $g(n)$ tale che la funzione $f(n) = n + 3n^2 + 4n^3$ sia $f(n) = \Omega(g(n))$. Si *dimostri* che la funzione scelta soddisfi il requisito.

6. (a) **Dimostrare** che la funzione $f(n) = n \log n^2 + 2n + 3$ è $O(n^2)$.

   (b) **Dimostrare** che la funzione $f(n) = 2n^3 + n^2 \log n$ è $\Omega(n^2)$.

7. **Dire** quali delle seguenti quattro affermazioni sono vere (non è necessario giustificare la risposta):

   (a) $0.003n^3 + 982n^2 + 3n = O(n^2)$

   (b) $n = O(n^5)$

   (c) $n^2 = \Omega(n)$

   (d) $n^3 = \Omega(4n^3)$

8. **Dimostrare** che la funzione $f(n) = 7n^2 + 5n + 4$ è $\Theta(n^2)$.

9. Per ciascuna delle seguenti affermazioni, dire se essa è vera o falsa.

$7n^4 - 8n^3 + 5 = O(n^4)$	`Vero`
$7n^4 - 8n^3 + 5 = O(n^3)$	`Falso`
$7n^4 - 8n^3 + 5 = O(n^5)$	`Vero`
$7n^4 - 8n^3 + 5 = \Omega(n^4)$	`Vero`
$7n^4 - 8n^3 + 5 = \Omega(n^3)$	`Vero`
$7n^4 - 8n^3 + 5 = \Omega(n^5)$	`Falso`
$7n^4 - 8n^3 + 5 = \Theta(n^4)$	`Vero`
$7n^4 - 8n^3 + 5 = \Theta(n^3)$	`Falso`
$7n^4 - 8n^3 + 5 = \Theta(n^5)$	`Falso`

Esempi di analisi di algoritmi.

Tempo logaritmico: Il tempo di esecuzione dell'algoritmo è al più un fattore costante per il logaritmo della dimensione dell'input.

```
Algoritmo(n)
x = 0; y = 1
WHILE(y < n + 1){
    x = x + 1
    y = 2 × y
}
RETURN x
```

Osserviamo che dopo la prima iterazione del ciclo WHILE vale che $y = 2^1$, dopo la seconda iterazione vale che $y = 2^2$, dopo la terza iterazione vale che $y = 2^3, \ldots$, e così via. Pertanto, la iterazione i-esima in cui terminiamo sarà tale che $2^i \leq n < 2^{i+1}$, ovvero $i \leq \log n$. Poichè in ogni iterazione del ciclo WHILE eseguiamo un numero costante di operazioni, ne segue che la complessità $T(n)$ dell'algoritmo sarà $T(n) = O(\log n)$.

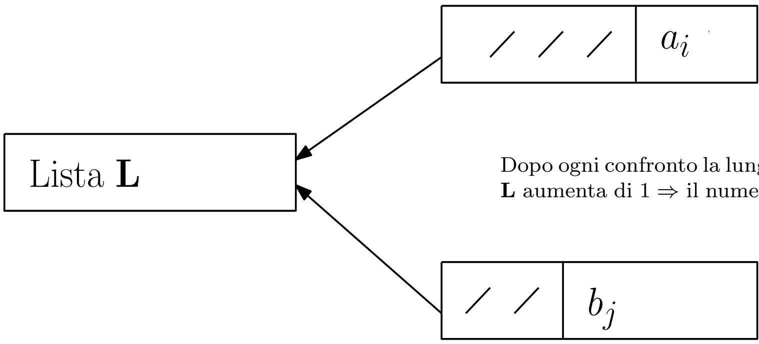
Tempo Lineare: Il tempo di esecuzione dell'algoritmo è al più un fattore costante per la dimensione dell'input.

Esempio: Calcolo del massimo di n numeri $a_1, a_2, \ldots, a_n$

```
max = a_1
FOR(i = 2; i < n + 1; i = i + 1){
    IF(a_i > max){
      max = a_i
     }
   }
RETURN max
```

Esempio: Merge. Trasforma due liste **ordinate** $A = a_1, \ldots, a_n$ e $B = b_1, \ldots, b_n$, in cui $a_1 \leq \ldots \leq a_n$ e $b_1 \leq \ldots \leq b_n$ in un'unica lista ordinata L.

```
Merge(A, B)
L = ∅
i = 1, j = 1
WHILE(entrambe le liste A, B non sono vuote){
    IF(a_i ≤ b_j){
      appendi a_i alla lista L, i = i + 1
      ELSE appendi b_j alla lista L, j = j + 1
   }
appendi il resto della lista non vuota, tra A e B, ad L
```



Dopo ogni confronto la lunghezza della lista output **L** aumenta di 1 $\Rightarrow$ il numero di operazioni $= O(n)$

Tempo quadratico. L'algoritmo esamina tutte le coppie di dati elementi

Esempio: Dati n punti, di coordinate $(x_1, y_1), \ldots, (x_n, y_n)$, si vuole determinare la coppia di punti più vicina

```
1. min = (x_1 - x_2)^2 + (y_1 - y_2)^2
2. FOR(i = 1, i < n + 1; i = i + 1){
3.     FOR(j = i + 1, j < n + 1; j = j + 1){
4.        d = (x_i - x_j)^2 + (y_i - y_j)^2
5.        IF(d < min){
6.        min = d
       }
    }
}
RETURN min
```

Analisi: il FOR delle linee 3.–5. esegue la prima volta $c(n-1)$ operazioni, la seconda volta $c(n-2)$ operazioni, la terza volta $c(n-3)$ operazioni, ... In totale, l'algoritmo esegue $c(n-1) + c(n-2) + c(n-3) + \ldots + c \cdot 1 = c \sum_{k=1}^{n-1} k = O(n^2)$ operazioni

Tempo cubico : $O(n^3)$

L'algoritmo esamina tutte le triple di dati elementi

Esempio: Dati n insiemi $S_1, \dots, S_n$, ciascuno di essi sottoinsieme di $\{1, \dots, n\}$, esiste una coppia (S_i, S_j) tale che $S_i \cap S_j = \emptyset$?

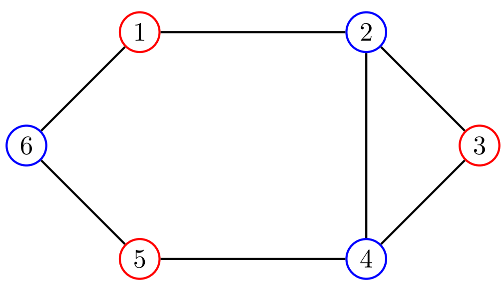
```
1. FOR(i = 1, i < n + 1, i = i + 1){
2.     FOR(j = i + 1, j < n + 1, j = j + 1){
3.       FOR(ogni elemento  x ∈ S_i){
4.           determina se x appartiene anche a S_j
         }
5.       IF(nessun elemento di S_i appartiene anche a S_j){
6.           RETURN( S_i ed S_j sono disgiunti)
     }
   }
7. RETURN(non esistono insiemi disgiunti)
```

Analisi: il FOR delle linee 3.–6 esegue $O(n)$ operazioni, il FOR delle linee 2.–6 esegue $O(n^2)$ operazioni,

In totale, l'algoritmo esegue $O(n^3)$ operazioni

Per introdurre l'ultimo esempio, diamo la seguente definizione. Un insieme di punti è detto **indipendente** se *nessuna* coppia di suoi elementi è unita da archi.

L'insieme $\{1, 3, 5\}$ è indipendente, l'insieme $\{2, 4, 6\}$ **non** è indipendente,



Problema: trovare il più grande insieme indipendente in un grafo con n punti (vertici).

Tempo esponenziale : $O(c^n)$

L'algoritmo esamina tutte le possibili soluzioni per trovare la "migliore"

1. $S^* = \emptyset$
2. **for** `ogni sottoinsieme di vertici` S
3. `controlla se` S `è indipendente`
4. **if** (S è il sottoinsieme indipendente più grande trovato finora)
5. `aggiorna` $S^* = S$
6. **return**(S^*)

Analisi: il **for** delle linee 2.–5. viene eseguito 2^n volte (tanti sono tutti i sottoinsiemi di n vertici),

controllare se un sottoinsieme S è indipendente richiede $O(n^2)$ operazioni (per ogni coppia di vertici in S occorre controllare se vi è un arco tra di loro).