

# Algoritmi

UniVR - Dipartimento di Informatica

**Fabio Irimie**

1° Semestre 2024/2025

## Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Confronto tra algoritmi . . . . .	2
1.2	Rappresentazione dei dati . . . . .	2
<b>2</b>	<b>Calcolo della complessità</b>	<b>2</b>
2.1	Linguaggi di programmazione . . . . .	2
2.1.1	Blocchi iterativi . . . . .	3
2.1.2	Blocchi condizionali . . . . .	3
2.1.3	Blocchi iterativi . . . . .	3
2.2	Esempio . . . . .	4
2.3	Ordine di grandezza . . . . .	5

# 1 Introduzione

Un algoritmo è una sequenza **finita** di **istruzioni** volta a risolvere un problema. Per implementarlo nel pratico si scrive un **programma**, cioè l'applicazione di un linguaggio di programmazione, oppure si può descrivere in modo informale attraverso del **pseudocodice** che non lo implementa in modo preciso, ma spiega i passi per farlo.

Ogni algoritmo può essere implementato in modi diversi, sta al programmatore capire qual'è l'opzione migliore e scegliere in base alle proprie necessità.

## 1.1 Confronto tra algoritmi

Ogni algoritmo si può confrontare con gli altri in base a tanti fattori, come:

- **Complessità**: quanto ci vuole ad eseguire l'algoritmo
- **Memoria**: quanto spazio in memoria occupa l'algoritmo

## 1.2 Rappresentazione dei dati

Per implementare un algoritmo bisogna riuscire a strutturare i dati in maniera tale da riuscire a manipolarli in modo efficiente.

# 2 Calcolo della complessità

La complessità di un algoritmo mette in relazione il numero di istruzioni da eseguire con la dimensione del problema, e quindi è una funzione che dipende dalla dimensione del problema.

La **dimensione del problema** è un insieme di oggetti adeguato a dare un'idea chiara di quanto è grande il problema da risolvere, ma sta a noi decidere come misurare il problema.

Ad esempio una matrice è più comoda da misurare come il numero di righe e il numero di colonne, al posto di misurarla come il numero di elementi totali.

La complessità di solito si calcola come il **caso peggiore**, cioè il limite superiore di esecuzione dell'algoritmo.

## 2.1 Linguaggi di programmazione

Ogni linguaggio di programmazione è formato da diversi blocchi:

1. **Blocco iterativo**: un tipico blocco di codice eseguito sequenzialmente e tipicamente finisce con un punto e virgola.
2. **Blocco condizionale**: un blocco di codice che viene eseguito solo se una condizione è vera.
3. **Blocco iterativo**: un blocco di codice che viene eseguito ripetutamente finché una condizione è vera.

Questi sono i blocchi base della programmazione e se riusciamo a calcolare la complessità di ognuno di questi blocchi possiamo calcolare più facilmente la complessità di un intero algoritmo.

### 2.1.1 Blocchi iterativi

$$\begin{array}{l} I_1 \quad c_1(n) \\ I_2 \quad c_2(n) \\ \vdots \quad \vdots \\ I_l \quad c_l(n) \end{array}$$

Se ogni blocco ha complessità  $c_1(n)$ , allora la complessità totale è data da:

$$\sum_{i=1}^l c_i(n)$$

### 2.1.2 Blocchi condizionali

$$\begin{array}{l} \text{IF cond} \quad c_{cond}(n) \\ I_1 \quad c_1(n) \\ \text{ELSE} \\ I_2 \quad c_2(n) \end{array}$$

La complessità totale è data da:

$$c(n) = c_{cond}(n) + \max(c_1(n), c_2(n))$$

A volte la condizione è un test sulla dimensione del problema e in quel caso si può scrivere una complessità più precisa.

### 2.1.3 Blocchi iterativi

$$\begin{array}{l} \text{WHILE cond} \quad c_{cond}(n) \\ I \quad c_0(n) \end{array}$$

Si cerca di trovare un limite superiore  $m$  al limite di iterazioni.

Di conseguenza la complessità totale è data da:

$$c_{cond}(n) + m(c_{cond}(n) + c_0(n))$$

## 2.2 Esempio

### Esempio 2.1

Calcoliamo la complessità della moltiplicazione tra 2 matrici:

$$A_{n \times m} \cdot B_{m \times l} = C_{n \times l}$$

L'algoritmo è il seguente:

```
1  for i <- 1 to n // n ( 5 ml + 4l + 2 ) + n + 1
2
3  for j <- 1 to l // l ( 5m + 2 + 1 ) + 1 + l
4      c[i][j] <- 0
5      for k <- 1 to m // ( m + 1 + m(4) )
6          // 3 (moltiplicazione, somma e assegnamento)
7          // 1 (incremento for)
8          c[i][j] += a[i][k] * b[k][j]
9
```

Partiamo calcolando la complessità del ciclo *for* più interno. Non ha senso tenere in considerazione tutti i dati, ma solo quelli rilevanti. In questo caso avremo:

$$(m + 1 + m(4)) = 5m + 1$$

Questa complessità contiene informazioni poco rilevanti perchè possono far riferimento alla velocità della cpu e un millisecondo in più o in meno non cambia nulla se teniamo in considerazione solo l'incognita abbiamo:

$$m$$

Questo semplifica molto i calcoli, rendendo meno probabili gli errori. Siccome la complessità si calcola su numeri molto grandi, le costanti piccole prima o poi verranno tolte perchè poco influenti.

La complessità totale alla fine sarebbe stata:

$$5nml + 4ml + 2n + n + 1$$

Ma ciò che ci interessa veramente è:

$$5nml + 4ml + 2n + n + 1$$

Se non consideriamo le costanti inutili, la complessità finale è:

$$nml$$

Nella maggior parte dei casi ci si concentra soltanto sull'ordine di grandezza della complessità, e non sulle costanti.

## 2.3 Ordine di grandezza

L'ordine di grandezza è una funzione che approssima la complessità di un algoritmo:

$$f \in O(g)$$
$$\exists c > 0 \exists \bar{n} \forall n \geq \bar{n} \quad f(n) \leq cg(n)$$

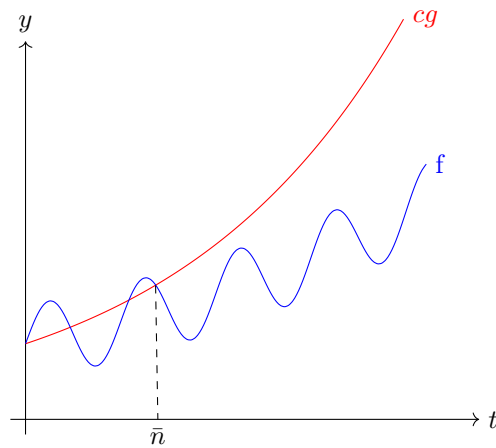


Figura 1: Esempio di funzione  $f \in O(g)$

$$f \in \Omega(g)$$

Da fare a casa