

Algoritmi Numerici e Strumenti di Calcolo

SUPSI Dipartimento Tecnologie Innovative

Gianni Grasso

29 settembre 2024

Classe: I1B

Anno scolastico: 2024/2025

Indice

1	Introduzione	3
2	Algoritmo di Horner	4
2.1	Valutazione di un polinomio	4
2.2	Applicazione	5
2.3	Horner inverso	5

1 Introduzione

La complessità (o efficienza) di un algoritmo è la quantità di risorse necessaria per la risoluzione di un determinato problema. Le risorse possono essere quantificate in termini di memoria o **tempo computazionale**.

La complessità temporale viene solitamente misurata contando il numero di operazioni elementari che devono essere svolte dall'algoritmo e viene espressa tramite la **notazione O-grande**, che si concentra sul termine di grado maggiore, escludendo i coefficienti e i termini di grado inferiore. Ad esempio, supponiamo che un determinato algoritmo richieda di svolgere $2n^3 + 4n^2 + n$, la sua complessità temporale è definita come $O(n^3)$ perchè il termine maggiore è n^3 .

Ecco altri esempi di complessità temporali, dalla minore alla maggiore:

- $O(1)$, tempo costante
- $O(\log n)$, tempo logaritmico
- $O(n)$, tempo lineare
- $O(n^2)$, tempo quadratico
- $O(n^3)$, tempo cubico

Facciamo un esempio pratico, scriviamo un algoritmo per sommare i primi n numeri interi partendo da 1. Il metodo più intuitivo sarebbe di rappresentarlo come:

$$S_n = \sum_{i=1}^n i = 1 + 2 + \dots + n$$

questo algoritmo richiede di svolgere $n - 1$ addizioni, la sua complessità temporale è quindi $O(n)$, ossia lineare.

Proviamo ora a trovare un algoritmo che faccia la medesima cosa ma con una complessità minore, consideriamo

$$S_n = \frac{n(n+1)}{2}$$

la cui validità può essere dimostrata matematicamente per induzione, non verrà dimostrata in questo documento.

Proviamo ad applicare questa seconda formula ai casi $n = 100$ e $n = 1000$:

$$S_{100} = \frac{100 + 101}{2} = 5050$$

$$S_{1000} = \frac{1000 + 1001}{2} = 500500$$

Notiamo che, in entrambi i casi, abbiamo dovuto effettuare solo tre operazioni distinte (un'addizione, una moltiplicazione e una divisione). Possiamo concludere che la complessità temporale di questo algoritmo è $O(1)$, ossia è costante e **indipendente dal valore di n** , a prescindere da quanto esso sia grande, preferiamo quindi usare il secondo algoritmo rispetto al primo.

2 Algoritmo di Horner

2.1 Valutazione di un polinomio

Per capire Horner dobbiamo prima fare degli step intermedi che dimostrano la sua effettiva utilità, consideriamo un generico polinomio di terzo grado:

$$P(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

dove a_0, \dots, a_3 sono coefficienti reali qualsiasi.

Calcoliamo quante operazioni sono necessarie per calcolare il valore del polinomio $P(x)$ in un generico punto x_0 effettuando le moltiplicazioni e le addizioni richieste:

$$P(x_0) = \underbrace{a_3x_0^3}_{3 \text{ molt.}} + \underbrace{a_2x_0^2}_{2 \text{ molt.}} + \underbrace{a_1x_0}_{1 \text{ molt.}} + a_0$$

notiamo che sono necessarie esattamente 6 moltiplicazioni e 3 addizioni.

Ora invece consideriamo un generico polinomio di grado n :

$$P(x) = \sum_{k=0}^n a_k x^k = a_0 + a_1x + \dots + a_nx^n$$

dove a_0, \dots, a_n sono coefficienti reali qualsiasi.

Abbiamo quindi lo stesso scenario del caso precedente ma con un polinomio di grado n invece che di grado 3. Procediamo in modo analogo a quanto fatto prima per determinare il numero di operazioni necessarie per calcolare il valore del polinomio x_0 :

$$P(x_0) = \underbrace{a_nx_0^n}_{n \text{ molt.}} + \underbrace{a_{n-1}x_0^{n-1}}_{n-1 \text{ molt.}} + \dots + \underbrace{a_1x_0}_{1 \text{ molt.}} + a_0$$

Sfruttando la formula introdotta nello scorso capitolo risulta che servono:

$$\underbrace{\frac{n(n+1)}{2}}_{\text{molt.}} + n = \frac{1}{2}n^2 + \frac{3}{2}n$$

operazioni, che corrispondono a una complessità temporale **quadratica**, ossia $O(n^2)$, poiché il termine maggiore è n^2 .

Se ora consideriamo nuovamente il polinomio di terzo grado di prima possiamo notare che esso può essere riscritto come:

$$P(x) = a_3x^3 + a_2x^2 + a_1x + a_0 = ((a_3x + a_2)x + a_1)x + a_0$$

la cui correttezza può essere verificata sviluppando i conti. Notiamo che il numero di operazioni è diminuito, prima erano 6 moltiplicazioni e 3 addizioni mentre con la nuova formula le moltiplicazioni sono 3.

Questa formula può essere generalizzata ad un polinomio di grado n :

$$P(x_0) = (((a_nx_0 + a_{n-1})x_0 + a_{n-2}) + \dots)x_0 + a_0$$

che corrisponde all'**algoritmo di Horner**, che consente di risolvere un polinomio mediante n moltiplicazioni e n addizioni (come visto prima). In totale sono quindi necessarie $2n$ operazioni e ciò corrisponde a una complessità temporale **lineare**, ossia $O(n)$.

2.2 Applicazione

L'algoritmo di Horner è utile per passare da un sistema di numerazione qualsiasi al sistema decimale (ad esempio da binario a decimale).

Se consideriamo ad esempio il numero 3152, possiamo scriverlo come:

$$\begin{aligned} 3152 &= 3 \cdot 1000 + 1 \cdot 100 + 5 \cdot 10 + 2 \\ &= 3 \cdot 10^3 + 1 \cdot 10^2 + 5 \cdot 10^1 + 2 \cdot 10^0 \end{aligned}$$

A ciascuna cifra viene attribuita una diversa potenza di 10 in base alla posizione che occupa all'interno del numero considerato. Un generico numero $x \in \mathbb{N}$ può essere scritto come:

$$x = a_n 10^n + a_{n-1} 10^{n-1} + \dots + a_0 = \sum_{i=0}^n a_i 10^i$$

dove a_0, \dots, a_n corrisponde alle cifre del numero.

Possiamo notare che la scrittura riportata è analoga a quella vista nella dimostrazione, che possiamo quindi riscrivere il numero come:

$$x = (((a_n \cdot 10 + a_{n-1}) \cdot 10 + a_{n-2}) + \dots) \cdot 10 + a_0$$

Per quanto riguarda gli altri sistemi numerici vale lo stesso discorso, consideriamo quindi un numero in una qualsiasi base b e scriviamolo nel corrispondente in base 10:

$$x = a_n b^n + a_{n-1} b^{n-1} + \dots + a_0 = \sum_{i=0}^n a_i b^i$$

Ad esempio se vogliamo convertire 1201_3 in base 10, svolgiamo questi calcoli:

$$\begin{aligned} 1201_3 &= \sum_{i=0}^3 a_i 3^i \\ &= 1 \cdot 3^3 + 2 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 \\ &= 27 + 18 + 1 = 46_{10} \end{aligned}$$

Che utilizzando l'algoritmo di Horner diventa:

$$\begin{aligned} 1201_3 &= ((1 \cdot 3 + 2) \cdot 3 + 0) \cdot 3 + 1 \\ &= (5 \cdot 3 + 0) \cdot 3 + 1 \\ &= 15 \cdot 3 + 1 = 46_{10} \end{aligned}$$

2.3 Horner inverso

L'algoritmo di Horner inverso, o **algoritmo del modulo**, consente invece di passare da base 10 a base b .