Università di Trieste

Laurea in ingegneria elettronica e informatica

Enrico Piccin - Corso di Algoritmi e Strutture dati - Prof. Andrea Sgarro

Anno Accademico 2021/2022 - 2 Marzo 2022

Indice

2 Marzo 2022

1 Introduzione

Algoritmo è una parola molto antica, non connessa all'utilizzo e all'invenzione del calcolatore.
Liber abaci (tradotto "Libro della computazione"), scritto nel 1200 da Leonardo Bonacci, in contatto con la popolazione araba, in quanto mercante ed è venuto a conoscenza della **numerazione araba**, introducendola in Occidente e spiegandolo all'interno del **Liber abaci**.
La numerazione araba è uno straordinario passo in avanti nella scienza, in quanto viene introdotto il concetto di **notazione posizionale**, così come l'importanza del numero 0: i numeri non servono solamente per contare.
A Firenze, sempre negli stessi anni, ci fu una **protesta sindacale** contro l'innovazione tecnologica, contro una nuova scoperta, facendo pressione affinché il governo abolisse il nuovo sistema di numerazione, in quanto avrebbe fatto perdere il posto di lavoro a tutti coloro che prima eseguivano difficili calcoli con la numerazione romana: tuttavia, tale proteste possono rallentare il progresso, ma mai arrestarlo.
Leonardo Bonacci, nei suoi viaggi in Oriente, venne a conoscenza del **Liber abaci** di Al-Gorasmy, proveniente dalla Coresmia, ma che parlava persiano, da cui poi sarebbe stato tratto il nome **Algoritmo**, che letteralmente significa **procedimento di calcolo**.
Tuttavia, bisogna chiarire che algoritmo è un procedimento di calcolo non necessariamente numerico, ma molto più generale, che va ben al di là dei numeri.
Un altro importante elemento che contraddistingue l'algoritmo è òa **meccanicità**, ovvero la sua esecuzione può essere affidata ad una macchina: non deve necessariamente essere meccanizzato, ma deve essere meccanizzabile; in altre parole, l'esecuzione (bada bene, l'esecuzione e non la sua ideazione) dell'algoritmo è completamente **stupida**.

Esempio 1: L'algoritmo della moltiplicazione è molto chiaro e semplice: basta solamente accedere ad una **base di dati** in cui sono memorizzati i prodotti elementari fra numeri molto piccoli, e quindi molto più semplici da trattare.
Una volta eseguite le operazioni di moltiplicazione tramite quanto esposto in precedenza, è necessario eseguire delle operazioni di addizione, che era necessario aver precedentemente memorizzato.
Ecco che quello che si è appena eseguito è un **procedimento di calcolo**.

Esempio 2: L'algoritmo della divisione fa sempre uso di una base di dati, nella quale devono essere memorizzati i risultati del prodotto del divisore con tutti i numeri decimali da 0 a 9 e confrontare ciascun prodotto con il termine da dividere per ottenere il quoziente.
Alternativamente, si sarebbe potuto creare un ciclo da 0 a 9 in cui per ogni indice si sarebbe dovuto verificare se questo fosse il fattore moltiplicativo corretto per ottenere la quantità corretta da sottrarre.

Osservazione: In ciascuno di tali esempi è essenziale la meccanicità del processo esecutivo, che appare evidente.

Quando si rappresentano delle quantità e, a maggior ragione, quando si effettuano dei calcoli, è fondamentale fissare una base di rappresentazione, da cui poi dipendono le cifre che si possono impiegare per la rappresentazione.
La notazione posizionale permette anche di comprendere la rappresentazione di qualsiasi in quantità con qualsiasi base, effettuando anche delle conversioni di base a seconda della maggiore o minore convenienza di rappresentazione.
Per esempio, volendo convertire una quantità rappresentata in base $\mathcal{B} = 7$ in una base $\mathcal{C} = 10$ si deve procedere come segue

$$(5203)_7 = 5 \cdot 7^3 + 2 \cdot 7^2 + 0 \cdot 7^1 + 3 \cdot 7^0 = 1715 + 98 + 0 + 3 = (1816)_{10}$$

Ovviamente le basi di rappresentazione sono almeno binarie, in quanto la **base unaria** non può, per ovvie ragioni, non permette di rappresentare alcuna quantità se non quella unica che viene permessa dalla base scelta, ossia lo 0.
Tuttavia, il processo inverso, atto a passare dalla rappresentazione di una quantità in base 10 ad una in base 3, non risulta essere così immediato.

Per cercare un algoritmmo che permette di effettuare tale conversione, si effettua un primo passaggio controintuitivo, che prevede di rappresentare una quantità in base 10 in una quantità ancora in base 10, tramite un processo di divisioni successive. Si consideri, a tal proposito

$$(3412)_{10}$$

e si divida progessivamente tale numero per 10, come segue

3412	10
341	2
34	1
3	4
0	3

Leggendo, ora, i resti, al contrario si ottiene il numero cercato all'inizio. Se ora si prova a considerare un'altra base, come 3, l'operazione porta ad un risultato analogo

3412	3
1137	1
379	0
126	1
42	0
14	0
4	2
1	1
0	1

Per cui si è ottenuto

$$(3412)_{10} = (11200101)_3$$

Scegliendo la base 2 si ottiene, per esempio

241	2
120	1
60	0
30	0
15	0
7	1
3	1
1	1
0	1

Per cui si è ottenuto

$$(241)_{10} = (11110001)_2$$

Ovviamente la lunghezza di rappresentazione in base 2 prevede un numero di cifre pari a circa il triplo di quelle impiegate per rappresentare la medesima quantità in base 10, proprio perché

$$\log_2(10) \cong 3.3$$

Per passare da base 10 a base 100, le operazioni sono molto semplici

$$(375712)_{10} = [(37)\,(57)\,(12)]_{100}$$

usando come simboli

$$(00)\,,(01)\,,...,(75)\,,...,(99)$$

Si consideri, ora la base 8 e si scriva un numero binario in base ottale:

$$(010101010)_2 = [(010)\,(101)\,(010)]_8 = (252)_8$$

Ancora una volta, le cifre impiegate per la rappresentazione sono state ridotte ad un terzo, sempre perché

$$\log_2(8) = 3$$

E se ora si volesse impiegare la base 16 si otterrebbe:

$$(010101010)_2 = [(1010)\,(1010)]_{16} = (\mathrm{AA})_{16}$$

Osservazione: Si consideri una lunghezza $l = 5$. Allora usando 5 cifre, non tutte nulle, in base 10, i numeri n che si possono rappresentare sono

$$10000 \leq n \leq 99999 \quad \equiv \quad 10^4 \leq n < 10^5 \quad \equiv \quad 10^{l-1} \leq n < 10^l$$

Da ciò si può estrapolare un risultato importante

$$\log_{10}\left(10^{l-1}\right) \leq \log_{10}(n) < \log_{10}\left(10^l\right) \quad \equiv \quad l - 1 \leq \log_{10}(n) < l$$

che è una relazione esatta. Tuttavia, approssimativamente, si può scrivere che

$$l_{10}(n) \cong \log_{10}(n)$$

3 Marzo 2022

Com'è noto, il matematico indiano **Ramanujan** ha affermato che la matematica esatta non rappresenta una base solida per la realtà, mentre la matematica vera è fatta di approssimazioni.
Hardy scoprì quanto fosse importante lo studio di **Ramanujan** e insieme a lui portò avanti la teoria dei numeri, una teoria **asintotica** che, come lui stesso affermava, non può essere esatta, ma fatta di approssimazioni.
Se, per esempio, si considera una quantità scritta in base 5, quale $n = (5412)_5$

$$(541)_5 = 5 \cdot 5^2 + 4 \cdot 5^1 + 1 \cdot 5^0 = (146)_{10}$$

Se, ora, si fissa una lunghezza $l = 4$, una lunghezza rigida, senza considerare zeri in testa, si può capire che con 4 cifre si possono rappresentare numeri nell'intervallo

$$1000 \leq n < 10000$$

ovvero tale per cui

$$5^{l-1} \leq n < 5^l$$

e ciò funziona con qualsiasi base, per cui, in generale, fissata una lunghezza l e una base $\mathcal{B}$ si ha che le quantità che possono essere rappresentate con l cifre, non tutte uguali a 0 è

$$\mathcal{B}^{l-1} \leq n < \mathcal{B}^l$$

Traducendo tale risultato tramite il logaritmo in base $\mathcal{B}$, sfruttando la crescenza in senso stretto della funzione logaritmica, si ottiene, equivalentemente

$$l - 1 \leq \log_{\mathcal{B}}(n) < l$$

in cui, ovviamente,

$$\log_{\mathcal{B}}(n) < l \leq \log_{\mathcal{B}}(n) + 1$$

che si può scrivere che

$$l_{\mathcal{B}}(n) \cong \log_{\mathcal{B}}(n)$$

... continua ... lunghezza analogica e digitale. Per esempio, si può osservare che

$$\log_2(10) \cong 3.38$$

per cui la lunghezza in base 2 è circa tre volte la lunghezza in base 10.

Esempio: Per trasformare un numero da base 10 in base 5 si deve procedere per divisioni successive per 5, considerando i resti (per questo si parla di *divisione intera*). Per esempio si ha che

$$10 \div 3 = 3 \text{ con resto di } 1$$

in cui, ovviamente, il resto r può essere

$$0 \leq r < D$$

con D divisore. Convertendo 32 da base 10 a base 5 ci si aspetta di ottenere un resto $0 \leq r \leq 4$.

1.1 Architettura dei calcolatori

Il calcolatore, naturalmente, si basa sulla logica binaria, ovvero opera impiegando la rappresentazione in base 2.
Il metodo più utilizzato per rappresentare caratteri diversi da quelli binari, tramite una codifica binaria, è il metodo ASCII (dall'inglese, American Standard Code For Information Interchange). Naturalmente, siccome la codifica tramite ASCII fa uso di soli 7 bit (sarebbero 8, ma un bit è riservato alla parità, per la rilevazione degli errori), il numero di n-uple binarie che si possono ottenere è 2^7, un numero certamente irrisorio per la rappresentazione di tutti i caratteri alfanumerici necessari per la comunicazione multilinugua.
In generale, fissata una lunghezza n, il numero di n-uple binarie distinte è, ovviamente, 2^n, che

rappresenta una crescita esponenziale, praticamente infinita, anche se, ovviamente, nella teoria sono ben limitati.

Esempio: Si consideri una macchina calcolatrice che considera delle istruzioni di 9 bit come di seguito esposto:

NOME ISTRUZIONE	ISTRUZIONE
ADD	$010 \times \times \times \times \times \times$
PUNCH	$100 \times \times \times \times \times \times$

Tabella 1: Tabella di istruzioni operative per un calcolatore

Naturalmente, tale linguaggio è **Assembly**, ovvero un linguaggio molto simile al linguaggio macchina, che risulta particolarmente complesso da impiegare per lo sviluppo di software.

1.2 Diagramma di flusso

Si consideri il seguente **flowchart**, o **diagramma di flusso**:

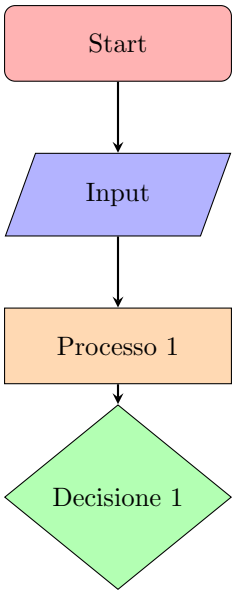

Figura 1: Diagramma di flusso

Tuttavia, tale tecnica di progettazione algoritmica è oramai superata, lasciando il posto allo **pseudocodice**, ossia un linguaggio di definizione delle istruzioni slegato da qualsiasi specifico linguaggio di programmazione di riferimento, che permette di esporre una serie di istruzioni esecutive molto simili a quelle di un programma vero e proprio.

2 Ordinamento (sorting)

Si espongono, di seguito, i principi di algoritmica dei più importanti algoritmi di ordinamento. Ciascuno di tali algoritmi prevede di effettuare l'ordinamento di n numeri forniti come input, in modo debolmente crescente, in caso di uguaglianza.

2.1 Bubble-Sort

Si espone di seguito l'algoritmo di ordinamento **bubble-sort** impiegando lo *pseudocodice*:

1. do the following $n-1$ times
   (a) point to the 1^{st} element
   (b) do the following $n-1$ times
      i. compare with next
      ii. if wrong order exchange
      iii. point to the next

Naturalmente tale algoritmo è corretto e lo si può verificare immediatamente, considerando, per esempio, i seguenti 5 elementi, così ordinati:

2	3	1	4	5

Ovviamente il procedimento ci porta ad eseguire l'algoritmo 4 volte. Nella prima iterazione si ottiene

2	1	3	4	5

la seconda iterazione, invece, porta ad ottenere

1	2	3	4	5

mentre le ultime due iterazioni sono superflue. Si capisce facilmente che tale algoritmo non risulta essere pienamente efficiente, in quante alcune iterazioni potrebbero essere evitate, tramite un **flag**, per esempio. Allo stato attuale, il numero delle iterazioni da eseguire è

$$\#\text{iterazioni} = (n-1) \cdot (n-1)$$

Se, invece, si facesse in modo di evitare alcune iterazioni si avrebbe un numero di iterazioni

$$(n-1) \leq \#\text{iterazioni} \leq (n-1)^2$$

considerando $n-1 \cong n$, e quindi $(n-1)^2 \cong n^2$ si può di re che la complessità del bubble-sort è **quadratica**, in quantoil numero delle iterazioni è n^2.