Algoritmi Numerici (Parte I)

[Lezione 1] Algoritmo di Horner e suo Inverso

Alessandro Antonucci
alessandro.antonucci@supsi.ch

https://colab.research.google.com/drive/1rjCcQMtkfmHeLEm593Ew3JlXm2dEoQSd

Algoritmi

- Somma dei primi **100** numeri? **5050**!
- Approccio lento $1 + 2 + 3 + \ldots + 98 + 99 + 100$
- Approccio veloce $(1 + 100) + (2 + 99) + \ldots = 101 \times 50$

Dato n *(input) calcolare somma* s *(output) primi* n *numeri?*

- Algoritmo "lento" esegue $n - 1$ addizioni (complessità $O(n)$, ovvero lineare)
- Algoritmo veloce $s = \frac{n(n+1)}{2}$ (1 somma, 1 molt , 1 div) (complessità $O(1)$, ovvero costante)

Linguaggi

- Linguaggio = sistema codificato di simboli che permette di esprimere un insieme di concetti
- Linguaggio numerico = sistema codificato di simboli che permette di esprimere gli elementi di un insieme numerico

Es. rappresentazione numeri (naturali) in base 10

Simboli = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, *Codice = sistema posizionale*

$$4273 = 4 \times 1000 + 2 \times 100 + 7 \times 10 + 3$$

$$4273 = 4 \times 10^3 + 2 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$$

Un algoritmo di lettura per i numeri in base 10

- Numero rappresentato come array (vettore) di digits (cifre)
  $k = (d_n, d_{n-1}, \ldots, d_1, d_0)$
- Formula di lettura $k = \sum_{i=0}^{n} d_i \times 10^i$
- Dalla formula al(lo pseudo) codice

```
output = 0
input = d(n),d(n-1), ... d(1),d(0)
for = 0 ... n
    output = output + d(i) * 10**i
end
return output
```

L’algoritmo di Horner (base 10)

- $4273 = ((4 \times 10 + 2) \times 10 + 7) \times 10 + 3$
- Schema di lettura più rapido

```
input = d(n),d(n-1), ... d(1),d(0)
output = d(n)
for i = n-1 ... 0
    output = output * 10 + d(i)
end
return output
```

- Complessità $O(n)$ (lineare) anziché quadratica!
- Funziona in qualunque base!

L'algoritmo di Horner (base b)

- $1302_4 = 1 \times 4^3 + 3 \times 4^2 + 0 \times 4^1 + 2 \times 4^0 = 114_{10}$
- $1302_4 = ((1 \times 4 + 3) \times 4 + 0) \times 4 + 2 = 114_{10}$
- $(d_n, d_{n-1}, \ldots, d_1, d_0) = \sum_{i=0}^{n} d_i \times b^i$

```
input = d(n),d(n-1), ... d(1),d(0)
output = d(n)
for i = n-1 ... 0
    output = output * b + d(i)
end
return output
```

- L'algoritmo converte in base 10 un numero in base b

L'operatore modulo

- $n \mod m$ è il resto della divisione intera di n per m
- sottraggo m a n finché non ottengo un numero $< m$,
- $n \mod m$ è un numero compreso fra 0 e $m - 1$
- Es. $n \mod 2$ è 0 per n pari ed 1 per n dispari
- Es. $n \mod 3$ è 0 per n multiplo di 3, 1 se $n - 1$ è multiplo di 3, 2 se $n + 1$ è multiplo di 3
- Es. $n \mod 10$ è la cifra più a destra di n

L'inverso dell'algoritmo di Horner

- Convertire in base $\boldsymbol{b}$ un numero in base $\mathbf{10}$?
- Invertire l'algoritmo di Horner!
- $114_{10} = \ldots_4$?

$$
\begin{array}{ll}
114 \mod 4 = 2 & (114-2)/4 = 28 \\
28 \mod 4 = 0 & (28-0)/4 = 7 \\
7 \mod 4 = 3 & (7-3)/4 = 1 \\
1 \mod 4 = 1 & (1-1)/4 = 0
\end{array}
$$

- $114_{10} = 1302_4$!

Da base m a base 10

$$100110111_2 = 311_{10}$$

$$200102_3 = 497_{10}$$

$$2177_8 = 1151_{10}$$

$$AC12_{16} = 44050_{10}$$

$$1111_3 = 40_{10}$$

$$1111_4 = 85_{10}$$

Nota: un numero in base b usa le cifre $\{0, 1, 2, \ldots, b-1\}$

Se $b > 10$ servono nuove cifre (es. le lettere).
Esadecimale $b = 16$:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Da base 10 a base n

$$2143_{10} = 100001011111_2$$

$$13458_{10} = 3492_{16}$$

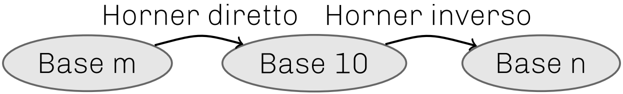
$$798_{10} = 1436_8$$

$$144_{10} = 10010000_2$$

$$144_{10} = 12100_3$$

$$144_{10} = 2100_4$$

Da base m a base n



- $1032_4 = 78_{10} = 2220_3$
- $101101_2 = 45_{10} = 140_5$
- $2177_8 = 1151_{10} = 47F_{16}$

Da base m a base n (trasformazioni dirette)

$210745_{10} = \ldots_{100}?$

$2 \cdot 10^5 + 1 \cdot 10^4 + 0 \cdot 10^3 + 7 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0 =$

$21 \cdot 100^2 + 7 \cdot 100^1 + 45 \cdot 100^0$

$\boxed{2}\boxed{1}\boxed{0}\boxed{7}\boxed{4}\boxed{5}$

$\boxed{21} \quad \boxed{7} \quad \boxed{45}$

$100 = 10^2$: cifre raggruppate a due a due

Stessa cosa se base b e non 10

Da binario ad hexa (e viceversa)

- $1032_4 = 1001110_2$
- $101101_2 = 2D_{16}$
- $2177_8 = 010001111111_2$

Esercitazione (Esercizio 1)

Eseguire i seguenti cambiamenti di base

- $\mathbf{2177_8} = \ldots_2 (= 1151_{10} = 10001111111_2)$
- $\mathbf{10220_3} = \ldots_9 (= 105_{10} = 126_9)$
- $\mathbf{20016_7} = \ldots_5 (= 4815_{10} = 123230_5)$
- $\boldsymbol{AC12_{16}} = \ldots_8 (= 44050_{10} = 126022_8)$

Esercitazione (Esercizio 2)

Eseguire i seguenti cambiamenti di base

- $1111_2 = 15_{10}$, $11111_2 = 31_{10}$, $111111_2 = 63_{10}$
- $222_3 = 26_{10}$, $2222_3 = 80_{10}$, $22222_3 = 242_{10}$,
- $44_5 = 24_{10}$, $444_5 = 124_{10}$, $4444_5 = 624_{10}$,

$$\underbrace{bbbb \dots bb_{b+1}}_{\text{n cifre}} = (b+1)^n - 1$$

Esercitazione (Esercizio 3)

In C una variabile **`unsigned int`** richiede al compilatore di allocare 32 bit, che verranno utilizzati per rappresentare numeri interi senza segno secondo le regole del sistema posizionale in base due

`unsigned short/long` fanno lo stesso con 16/64 bit.

- Rappresentare il numero 131'086 nei tre formati
- Identificare il più grande numero in ogni formato
- Leggere le sequenze (compattate)

$$\mathbf{A019} \ (= 40985_{10}) \ \texttt{unsigned short}$$
$$\mathbf{010C|7142} \ (= 17592642_{10}) \ \texttt{unsigned int}$$
$$\mathbf{0000|100A|0007|3501} \ (\simeq 1.765 \cdot 10^{13}) \ \texttt{unsigned long}$$