Laboratorio di programmazione

17 gennaio 2005

Righello

Un "righello" con una tacca centrale lunga n può essere pensato come l'accostamento di due righelli con una tacca centrale lunga $n-1$, separati appunto da una tacca lunga n. Ad esempio, un righello con tacca centrale lunga 4 può essere stampato in modo rudimentale come segue:

```
-
- -
-
- - -
-
- -
-
- - - -
-
- -
-
- - -
-
- -
-
```

dove ogni tacca è stata rappresentata da n ripetizioni del carattere - (meno). Ovviamente, un righello con tacca centrale lunga 1 è costituito dalla sola tacca centrale.

Scrivete una funzione ricorsiva che, avendo come parametro la lunghezza della tacca centrale, stampi il relativo righello (come nell'esempio precedente).

Conversioni

Il problema di ottenere le cifre (della rappresentazione in una data base) di un numero naturale assegnato può essere risolto molto facilmente ottenendo dapprima le cifre della parte più significativa del numero e quindi la sola cifra meno significativa.

Ad esempio, le cifre del numero 123 in base dieci sono le cifre della rappresentazione di 12 seguite dalla cifra 3.

Più in generale, le cifre di x in base b sono date dalle cifre di x/b (dove $/$ rappresenta la divisione intera) seguite dalla cifra $x\%b$ (dove % rappresenta il resto della divisione intera).

Scrivete una funzione ricorsiva che, avendo come parametro due numeri naturali x e $b > 1$, stampi le cifre di x in base b.

Potenza

L'elevamento all'n-esima potenza di un numero x può essere ottenuto banalmente tramite n moltiplicazioni: $x \cdot x \cdots x$.

È possibile risparmiare molte moltiplicazioni calcolando la potenza tramite successivi elevamenti al quadrato.

Se, ad esempio, $n = 8$, si può scrivere $x^8 = ((x^2)^2)^2$, che può essere calcolato con tre moltiplicazioni come $a = x \cdot x$, $b = a \cdot a$, $c = b \cdot b$. Ossia l'esponente viene dimezzato ad ogni passaggio: $c = x^{8/2} \cdot x^{8/2}$, $b = x^{4/2} \cdot x^{4/2}$ e $a = x^{2/2} \cdot x^{2/2}$.

In generale non è sempre possibile dimezzare l'esponente, ma vanno considerati i casi in cui n è pari, o dispari:

$$x^n = \begin{cases} (x^k) \cdot (x^k) & \text{se } n = 2k, \text{ oppure} \\ x \cdot (x^k) \cdot (x^k) & \text{se } n = 2k+1. \end{cases}$$

dove per calcolare $(x^k) \cdot (x^k)$ è sufficiente calcolare x^k una sola volta (se $y = x^k$, allora $(x^k) \cdot (x^k) = y \cdot y$).

Scrivere una funzione ricorsiva che, dati x e n come parametri, restituisca x^n tramite quadrature successive.

Modificare la funzione in modo tale che restituisca, invece della potenza, il numero di moltiplicazioni necessarie ad ottenere il risultato. Come cresce il numero di moltiplicazioni in funzione di n?

Numeri di Fibonacci

I numeri di Fibonacci furono introdotti da Leonardo di Pisa, detto Fibonacci, per descrivere la crescita di una popolazione di conigli (sebbene in modo idealizzato e non plausibile dal punto di vista biologico). Per calcolare il numero di coppie di conigli nella popolazione all'n-esimo mese, egli fece le seguenti assunzioni:

- al primo mese, c'è una sola coppia di conigli,
- le coppie neonate diventano fertili a partire dal secondo mese di vita,
- ogni mese le coppie fertili generano una nuova coppia ciascuna,
- i conigli non muoiono mai.

Se traduciamo queste ipotesi in formule, chiamato F_n il numero di coppie di conigli presenti all'n-esimo mese, avremo $F_1 = F_2 = 1$ e, per i mesi successivi, $F_n = F_{n-1} + F_{n-2}$, infatti ad ogni mese avremo le coppie del mese precedente, più le nuove coppie generate dalle coppie nate due mesi prima. Il numero di coppie di conigli cresce come $1, 1, 2, 3, 5, 8, 13, 21, \ldots$.

Scrivete una funzione ricorsiva che, avendo come parametro n, restituisca il valore di F_n.

Modificate la funzione perché restituisca, invece di F_n, il numero di volte che viene chiamata per calcolare tale valore.

Scrivete una versione del programma che non usi la ricorsione.

Suggerimento. Una soluzione iterativa elementare si può ottenere utilizzando un vettore nell'n-sima posizione del quale sia immagazzinato F_n. Per riempire tale vettore basta un semplice ciclo `for`. Ma, a ben guardare, ogni volta userete solo due posizioni del vettore, questo suggerisce una soluzione che utilizzi solo tre variabili (invece di un vettore con n posti), memorizzando solo F_n, F_{n-1} e F_{n-2}.