



## Problema 10

19 Dicembre 2016

### Descrizione

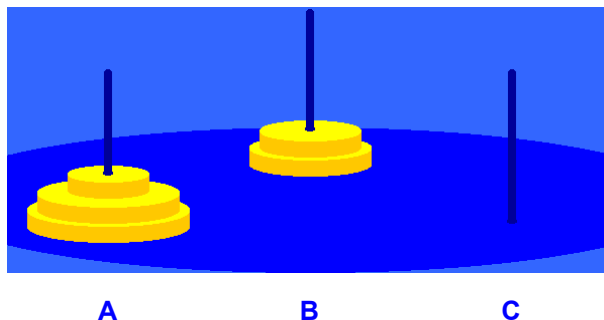
Il celebre rompicapo della *Torre di Hanoi* è costituito da tre asticelle (A, B e C), in una delle quali (A) sono inizialmente inanellati  $n$  dischi di diametro diverso e decrescente dal basso verso l'alto. Per risolvere il rompicapo si deve trasferire la torre di  $n$  dischi in corrispondenza a una diversa asticella (B), spostando un solo disco per volta da un'asticella ad un'altra e rispettando l'ulteriore vincolo che un disco non può mai essere appoggiato sopra un altro disco di diametro inferiore.

Il rompicapo ammette un'elegante soluzione ricorsiva: per trasferire una torre di  $n$  dischi dall'asticella  $s$  (sorgente) all'asticella  $d$  (destinazione) è necessario innanzitutto trasferire una torre di  $n-1$  dischi—tutti tranne quello alla base—dall'asticella  $s$  all'asticella  $t$  (transito), rispettando le regole del gioco; quindi si può spostare la base—il disco di diametro maggiore fra gli  $n$  considerati—da  $s$  a  $d$ ; infine ci si cimenta di nuovo con il sottoproblema di trasferire una torre di  $n-1$  dischi, nel secondo caso da  $t$  a  $d$ , sopra la base. Procedendo in questo modo sono necessarie  $2^n-1$  mosse per risolvere il rompicapo e si realizza così la strategia ottimale, che richiede il minor numero di mosse.

Dato un intero positivo  $n$ , la procedura `hanoi-moves` restituisce la sequenza di mosse, descritta da una stringa, che risolve il rompicapo della *Torre di Hanoi* di  $n$  dischi:

```
(define hanoi-moves ; valore: stringa
  (lambda (n)      ; n > 0 intero
    (hanoi-rec n "A" "B" "C")
  ))

(define hanoi-rec ; valore: stringa
  (lambda (n s d t) ; n intero, s, d, t: stringhe
    (if (= n 1)
        (string-append s "->" d)
        (let ((m1 (hanoi-rec (- n 1) s t d))
              (m2 (hanoi-rec (- n 1) t d s)))
          (string-append m1 " ", " s "->" d " ", m2)
        )
    )
  ))
```



'((1 4 5 A) (C) (2 3 B))

Nel caso di una torre iniziale di altezza  $n = 3$ , per esempio, il rompicapo viene risolto in 7 mosse, rappresentate dalla seguente stringa che risulta dall'applicazione della procedura `hanoi-moves`:

(hanoi-moves 3) → "A->B, A->C, B->C, A->B, C->A, C->B, A->B"

Il programma riportato sopra si basa sulla procedura `hanoi-rec` che applica una *ricorsione ad albero*. Tuttavia, per determinare una specifica mossa, la  $k$ -ima (dove  $1 \leq k \leq 2^n-1$ ) non è necessario utilizzare una ricorsione ad albero, ma è sufficiente una *ricorsione di coda*. A tal fine si può infatti osservare che, delle  $2^n-1$  mosse necessarie a risolvere il rompicapo, quella esattamente a metà, cioè la  $2^{n-1}$ -ima, permette di spostare il disco più grande. Per valori di  $k$  diversi da  $2^{n-1}$  ci si può quindi concentrare su uno solo dei due sottoproblemi che hanno come obiettivo trasferire una torre di  $n-1$  dischi, dove cambiano solo le asticelle sorgente e destinazione. Analogamente, non è necessario utilizzare la ricorsione ad albero per determinare una specifica configurazione del gioco, cioè la disposizione dei dischi nelle tre asticelle dopo la  $k$ -ima mossa.

Cerca di realizzare i programmi richiesti nei punti (i) e (ii) della pagina seguente *senza* usare la ricorsione ad albero.

(i) Definisci una procedura `hanoi-move` che, dati due interi positivi  $n$  e  $k$ , con  $k < 2^n$ , restituisce la  $k$ -ima mossa della sequenza (di mosse) che risolve il rompicapo della Torre di Hanoi di  $n$  dischi.

### Esempi

```
(hanoi-move 3 1)  →  "A->B"
(hanoi-move 3 2)  →  "A->C"
(hanoi-move 3 3)  →  "B->C"
(hanoi-move 3 4)  →  "A->B"
(hanoi-move 3 5)  →  "C->A"
(hanoi-move 3 6)  →  "C->B"
(hanoi-move 3 7)  →  "A->B"

(hanoi-move 5 6)  →  "C->B"
```

(ii) [La soluzione di questo problema è significativamente più impegnativa.] Definisci una procedura `hanoi-config` che, dati due interi  $n$ ,  $k$ , con  $n > 0$  e  $0 \leq k \leq 2^n - 1$ , restituisce la disposizione dei dischi al termine della  $k$ -ima mossa. Per  $k = 0$  la procedura restituisce la configurazione iniziale.

A tal fine conviene rappresentare gli  $n$  dischi con i numeri  $1, 2, \dots, n$ , dove a un numero più grande corrisponde un diametro maggiore. Una configurazione può allora essere descritta da una terna di liste (lista di tre liste), associate alle tre asticcioline, in cui i dischi sono ordinati per diametro crescente a partire dal primo elemento. Si suggerisce inoltre di non preoccuparsi che l'ordine delle tre liste nella terna sia predeterminato, ma di identificare l'asticciola a cui si riferisce con un elemento aggiuntivo alla fine di ciascuna lista della terna. In questo senso la configurazione illustrata dalla figura nella pagina precedente può essere descritta dalla terna `'((1 4 5 A) (C) (2 3 B))` come riportato in didascalia, oppure indifferentemente da `'((1 4 5 A) (2 3 B) (C))`, `'((C) (2 3 B) (1 4 5 A))`, ecc.

### Esempi

```
(hanoi-config 5 0)  →  '((1 2 3 4 5 A) (C) (B))
(hanoi-move 5 1)  →  "A->B"
(hanoi-config 5 1)  →  '((C) (1 B) (2 3 4 5 A))
(hanoi-move 5 2)  →  "A->C"
(hanoi-config 5 2)  →  '((1 B) (3 4 5 A) (2 C))
(hanoi-move 5 3)  →  "B->C"
(hanoi-config 5 3)  →  '((3 4 5 A) (1 2 C) (B))

(hanoi-config 5 5)  →  '((3 B) (1 4 5 A) (2 C))
(hanoi-move 5 6)  →  "C->B"
(hanoi-config 5 6)  →  '((1 4 5 A) (C) (2 3 B))

(hanoi-config 5 16) →  '((1 2 3 4 C) (5 B) (A))
(hanoi-config 5 31) →  '((C) (1 2 3 4 5 B) (A))
```