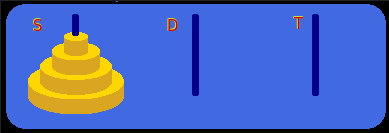
**HANOI**

****

(define hanoi-moves

(lambda (n) ; **N: numero dei dischi**

(hanoi-rec n 1 2 3) )) ;1 2 3 sono gli indici che individuano le aste **S D T**

;**NUMERO TOTALE DI MOSSE = 2^N**

;ritorna la **lista totale di mosse dalla prima all'ultima**

(define hanoi-rec ;ritorna la lista totale di mosse dalla prima all'ultima

(lambda (n s d t) ;source destination transit; n = numero dischi

(if (= n 1) ;rimasto 1 disco S --> D

(list (list s d)) ;MOSSA S --> D

;MOSSA 1 = sposta i primi **n-1 dischi da S --> T**

;MOSSA 2 = poi sposta **nesimo disco da S --> D**

;MOSSA 3 = per ultimo sposta **n-1 dischi da T --> D**

(let ( (m1 (hanoi-rec (- n 1) s t d) ) **;source = S destination = T**

(m2 (list s d)) ;**ultimo disco da S --> D**

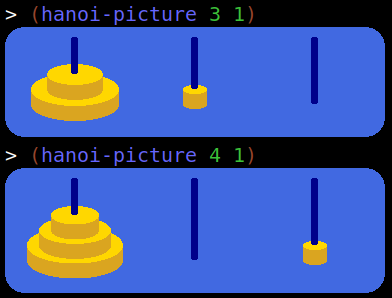
(m3 (hanoi-rec (- n 1) t d s)); **source = T, destination = D**

)

;; costruisce la lista delle **mosse in ordine da m1 a m3**

(append m1

(cons m2 m3) ) ))))



1. Configurazione dopo la **mossa 1 di hanoi-moves 3**
2. Condigurazione dopo la **mossa 1 di hanoi-moves 4**
3. Mossa 1 (hanoi-moves 4) == (hanoi-rec (- n 1) s t d) == **Mossa 1 (hanoi-rec 3)**
4. La **destinazione** del **sottoproblema** = **Mossa 1 (hanoi-moves 4) è T,**
   1. per spostare il blocco(n) più grande, tutti gli altri(n-1) devono essere spostati in T
   2. D ora è libero per poter contenere n-esimo blocco = MOSSA 2
5. **MOSSA 3 di (hanoi-moves 3): (n-1)blocchi** **da T** devono essere **spostati in D**
   1. La **destinazione** rimane la **stessa = D**, mentre il **punto** di **partenza** è **T**, invece di S

Immagine che contiene testo

Descrizione generata automaticamente

Alla **mossa intermedia[2^(n-1)] di (hanoi-moves n)**  si ha questa situazione…

**Perchè 2^(n-1) ???**

**Quante mosse** servono per **spostare (n-1)blocchi da S a T?**

**= 2^(n-1)**

;; Restituisce la **configurazione al termine della "k"-esima mossa**

(**define hanoi-disks** ; hanoi-disks: lista delle asticelle con il relativo numero di blocchi piazzati

(lambda (n k) ; **n = numero dischi**, **k = mossa inclusa** a cui **terminare** la **ricerca**

(**hanoi-rec-cod** n k

;configurazioni iniziali delle **3 postazioni con 0 dischi**

'(1 0) ;SOURCE

'(2 0) ;DESTINATION

'(3 0) ;TRANSIT

n) )) ;**n backup**

**;ricorsione di CODA: si stabilisce la posizione dell n-esimo disco,**

; **e via via sia arriva al minore**

**(define hanoi-rec-cod**  ; hanoi-rec-cod: list

(lambda (n k s d t n\_backup) ; n, k, n\_backup: int ; s, d, t: lists

(let ( (l (expt 2 (- n 1))) ;lunghezza = 2^(n-1)

(h (+ (cadr s) (cadr d) (cadr t))) ) ;**somma i contatori delle posizioni**

(cond ((= h n\_backup)  **;mosse terminate** perchè h è la somma di tutti i pezzi spostati

(list s d t) )

((< k l) **;mosse precedenti a quella intermedia**

;**deve ancora trasferire tutti gli (n-1)blocchi da S a T = mossa 1**

(hanoi-rec-cod (- n 1) k

(list (car s) (+ (cadr s) 1)) **;il blocco n-esimo si troverà ancora nella S**

t d n\_backup) )

((not (< k l)) **;mossa successiva a quella intermedia**

(hanoi-rec-cod (- n 1)

(- k l) **;toglie la mossa intermedia**

t **;n-1 blocchi da T a D passando per S = mossa3**

(list (car d) (+ (cadr d) 1)) **;incrementa il numero di oggetti in D = mossa 2**

s

n\_backup) )))))

**Restituisce l’immagine della configurazione dopo k mosse di (hanoi-moves3)**

(define recursive-picture

(lambda (N n k s d t ns nd nt img) ;**N: numero torale dei dischi da usare** per la **dimensione delle figure tramite il teachpack**

;**n: dischi ancora da spostare,** progressivamente diminuisce fino a 0, quando tutti saranno stati piazzati

**;k: mosse da trovare**

;**s d t = posizioni; ns nd nt: numero di disci per ogni posizione**

;image : immagine generata in precedenza a cui sovrapporre la nuova

(if (= n 0) img ;**tutti I dischi sono stati piazzati**

(if (< k (expt 2 (- n 1)))

**;mossa precedente a quella intermedia**

;**deve ancora finire di spostare (n-1)blocchi da S a T,**

;oppure ha appena finito ma non si può sapere

(recursive-picture N (- n 1) k **;configurazione di (n-1) blocchi dopo la k-mossa**

s t d ;da S --> T

(+ ns 1) **;il n-esimo si trova in S**

nt nd

(above (disk-image (- n 1) N s ns) img) ) ;nuova immagine generata

**;viene piazzato l'n-esimo blocco nell'asticella S**

**;mossa successiva o uguale**

;**n-esimo è sicuramente stato spostato in D**

;--> **si procede con le mosse in ordine giusto**

;=**= (n-1)blocchi da T a D**

(recursive-picture N (- n 1) (- k (expt 2 (- n 1)))

t d s ;da T --> D

nt

(+ nd 1) ;n-esimo si trova in D

ns

(above (disk-image (- n 1) N d nd) img) ))))) ;nuova immagine generata

;restituisce la configurazione grafica della situazione dopo K mosse

(define **hanoi-picture**

(lambda (n k)

(recursive-picture n n k

1 2 3 ;; S D T

0 0 0 ;ns nd nt

(towers-background n) ))) ;imposta la dimensione dello sfondo