

Laboratorio 7



R. Ferrero

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

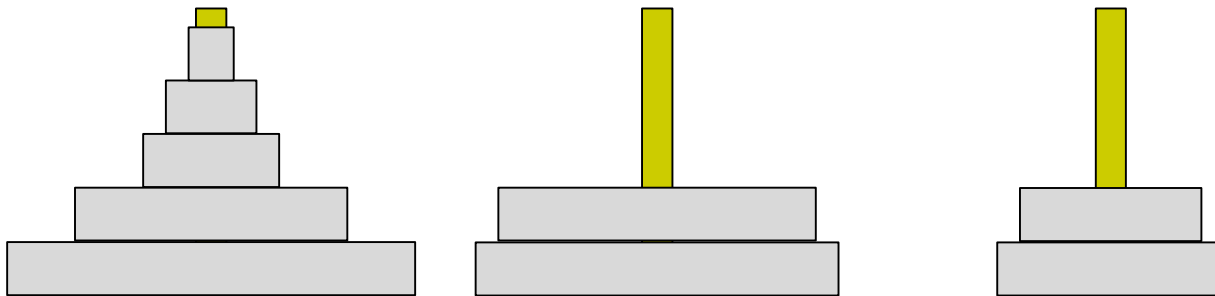
Torino - Italy

This work is licensed under the Creative Commons (CC BY-SA) License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>



Torre di Hanoi

- La torre di Hanoi è composta da 3 paletti.
- Ogni paletto contiene un certo numero di dischi (eventualmente zero), disposti in ordine di grandezza decrescente (il disco più grande è quello più in basso).



Torre di Hanoi: implementazione

- Ciascun paletto può essere rappresentato attraverso uno stack.
- Gli elementi dello stack indicano la dimensione dei dischi contenuti nel corrispondente paletto.
- I 3 stack sono definiti in un'area READWRITE: i 3 stack pointer puntano a locazioni di memoria differenti all'interno dell'area dati.
- E' lasciata libertà di scelta se gestire i 3 stack come full descending o empty ascending.

Esercizio 1

- Scrivere una subroutine `fillStack` per inizializzare lo stack associato ad un paletto.
- La subroutine riceve due parametri:
 - lo stack pointer associato al paletto
 - l'indirizzo ad un'area READONLY contenente una sequenza di costanti.
- La subroutine aggiorna i due parametri:
 - lo stack pointer punta all'ultimo dato inserito
 - l'indirizzo all'area READONLY è successivo all'ultima costante inserita.

Esercizio 1 (cont.)

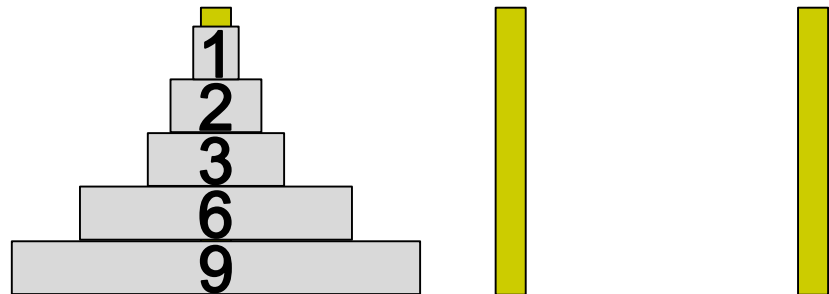
- I parametri sono passati alla subroutine attraverso lo stack principale (SP).
- L'inserimento termina al verificarsi di una fra le seguenti due condizioni:
 - la costante da inserire è 0
 - la costante da inserire è maggiore dell'ultimo elemento presente nello stack.

Esercizio 1: esempio

- $r1$ è lo stack pointer associato al primo paletto
- La sequenza di costanti è
DCD 9, 6, 3, 2, 1, 8, 7, 0, 5, 4, 0
- $r0$ contiene l'indirizzo della prima costante (9).
- Dopo aver chiamato `fillStack` la situazione è

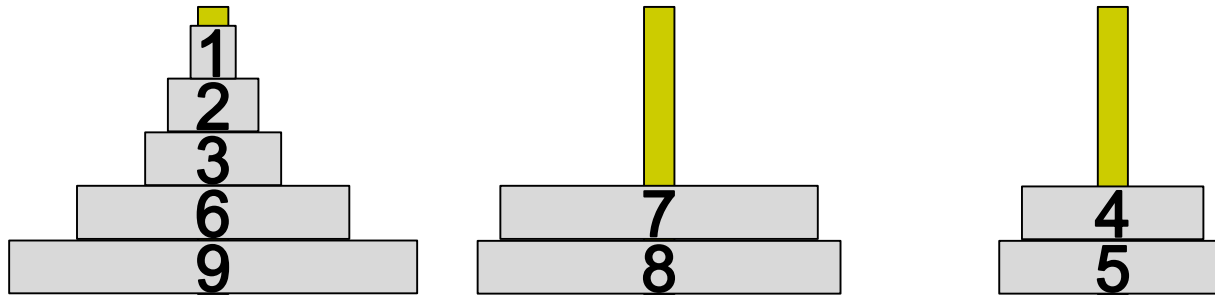
- $r1_{\text{new}} = r1_{\text{old}} \pm 20$

- $r0_{\text{new}} = r0_{\text{old}} + 20$



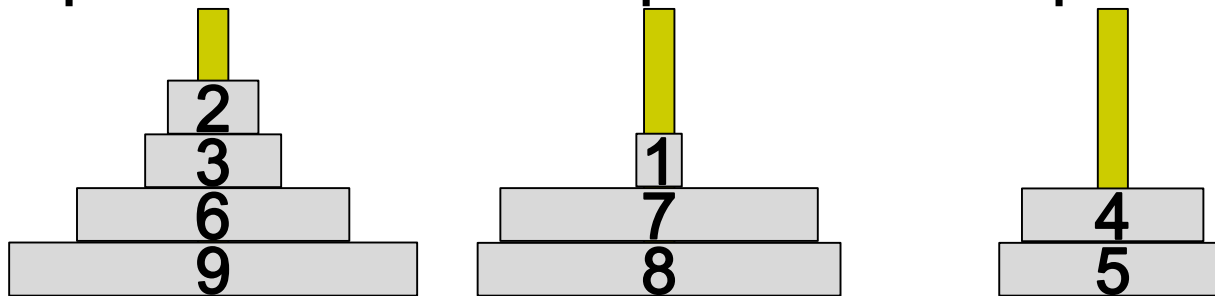
Esercizio 1: esempio

- Con tre chiamate consecutive a `fillStack` (una per ogni paletto) la torre di Hanoi diventa:



Torre di Hanoi: movimento singolo

- E' possibile spostare un disco da un paletto ad un altro se si verifica una delle due condizioni:
 - il nuovo paletto è vuoto
 - il disco superiore nel nuovo paletto ha dimensione maggiore rispetto al disco da spostare.
- Esempio: un disco è spostato dal paletto 1 al 2



Esercizio 2

- Scrivere una subroutine `move1` che gestisca lo spostamento di un disco.
- `move1` riceve tre parametri attraverso lo stack
 - stack pointer del paletto di partenza
 - stack pointer del paletto di destinazione
 - spazio per il valore di ritorno
- `move1` restituisce 1 se è stato possibile spostare il disco, 0 altrimenti. Gli stack pointer sono aggiornati se il disco è stato spostato.

Esercizio 2: avvertenza

- Per passare i parametri a `move1`, è preferibile usare 3 `PUSH` invece di una sola.
- Esempio: sia `r1` lo stack pointer del paletto di partenza, `r2` lo stack pointer del paletto di destinazione, `r0` contenga il valore di ritorno.
- `PUSH {r1, r2, r0}`
 `BL move1`
 `POP {r1, r2, r0}`
sposta dal paletto `r2` al paletto `r1`, perché i registri sono riordinati come `{r2, r1, r0}`

Esercizio 2: avvertenza

- Per garantire lo spostamento dal paletto $r1$ al paletto $r2$, si può usare il codice seguente:

```
PUSH { r1 }
```

```
PUSH { r2 }
```

```
PUSH { r0 }
```

```
BL move1
```

```
POP { r0 }
```

```
POP { r2 }
```

```
POP { r1 }
```

Torre di Hanoi: movimenti multipli

- E' possibile spostare N dischi dal paletto X al paletto Y usando una procedura ricorsiva:
 - Sposta i primi $N-1$ dischi da X a Z
 - Sposta il disco N da X a Y
 - Sposta $N-1$ dischi da Z a Y

Esercizio 3

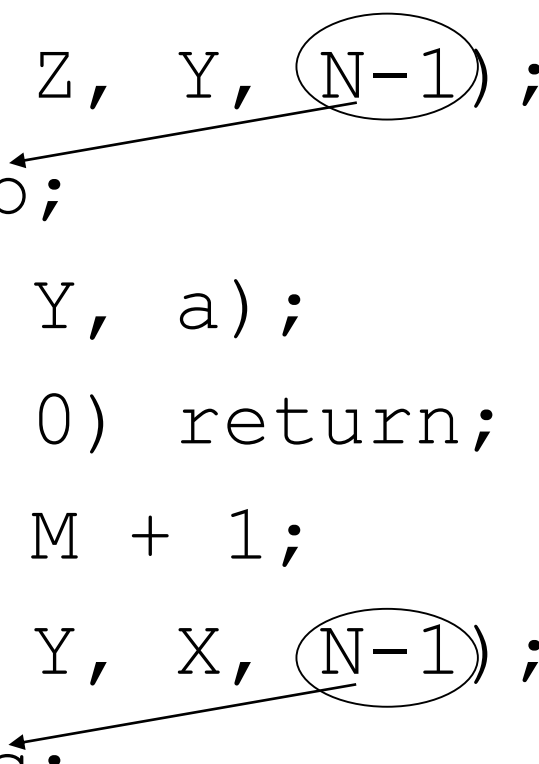
- Scrivere una subroutine `moveN` che gestisca lo spostamento di N dischi.
- `moveN` riceve attraverso lo stack:
 - stack pointer del paletto di partenza X
 - stack pointer del paletto di destinazione Y
 - stack pointer del paletto ausiliario Z
 - numero di dischi da spostare
- `moveN` aggiorna gli stack pointer e sostituisce il quarto parametro con il numero di movimenti effettuati.

Pseudocodice di moveN(X, Y, Z, N)

```
/* M è il num. di movimenti fatti*/  
M = 0;  
if (N == 1)  
{  
    move1(X, Y, a);  
/*a è il valore di ritorno: 0-1 */  
    M = M + a;  
}
```

Pseudocode di moveN(X, Y, Z, N)

```
else {  
    moveN(X, Z, Y, N-1);  
    M = M + b;  
    move1(X, Y, a);  
    if (a == 0) return;  
    else M = M + 1;  
    moveN(Z, Y, X, N-1);  
    M = M + c;  
}
```

A diagram illustrating the recursive nature of the pseudocode. It features two arrows. The first arrow originates from the circled 'N-1' in the first 'moveN' call and points to the variable 'b' in the assignment 'M = M + b;'. The second arrow originates from the circled 'N-1' in the second 'moveN' call and points to the variable 'c' in the assignment 'M = M + c;'. This visualizes how the problem size is reduced by one in each recursive call.

Esercizio 3: suggerimenti

- Siccome ogni chiamata recursiva aggiunge i propri parametri nello stack, si suggerisce di aumentare la dimensione dello stack modificando il valore di `Stack_Size`.
- Come verifica finale, considerando una torre di Hanoi con N dischi in un paletto e 0 negli altri due, constatare che il numero di movimenti necessari per spostare gli N dischi in un altro paletto è pari a $2^N - 1$.