Appunti senza pretese di P2+Lab:

Pensare ricorsivamente, programmare iterativamente

Alessandro Panconesi
DSI, La Sapienza
via Salaria 113, piano terzo
00198 Roma

In queste dispense ci occuperemo di un semplice rompicapo. Nel corso della soluzione scopriremo diverse cose interessanti.

1 Le Torri di Hanoi

In un monastero buddhista di Hanoi ci sono tre pali detti della terra, del mare e del cielo e che noi contrassegneremo piú prosaicamente come A, B e C. Inizialmente sul palo A era collocata una torre a forma di cono formata da 64 dischi concentrici di dimensioni tra loro tutte diverse. Ogni giorno un monaco compie il rituale di spostare uno dei dischi da un palo all'altro con lo scopo finale di spostare tutta la torre dal palo iniziale A al palo finale C. Nello spostamento dei dischi si deve sottostare a questa regola:

UN DISCO NON PUÓ MAI ESSERE POGGIATO SU UNO PIÚ PICCOLO.

Giorno dopo giorno, i dischi vengono spostati. Secondo la leggenda, il giorno in cui la torre sará completamente spostata dal primo all'ultimo palo il mondo finirá. Come vedremo, anche se ció fosse vero, non ci sarebbe molto di cui preoccuparsi. Per prima cosa sinceriamoci del fatto che spostare la torre da A a C é effettivamente possibile.

Fatto 1 *Esiste un metodo per spostare una torre di n dischi da A a C il cui numero di mosse é dato dalla formula*

$$T(n) = 2T(n-1) + 1.$$

con condizione iniziale $T(1) = 1$.

Dimostrazione: La dimostrazione é per induzione. Se $n = 1$ é certamente possibile spostare il disco da A a C per cui, denotando con $T(m)$ il numero di mosse necessarie per una pila di m dischi, si ha

$$T(1) = 1$$

e la base é provata. Veniamo al passo induttivo. Per chiarezza denotiamo i dischi come $1, 2, \ldots, n$ con i che denota non solo il disco i-esimo ma anche la sua larghezza (per cui 1 é il piú piccolo ed n il piú grande). L'idea é la seguente. Assumiamo per induzione che l'asserto sia vero per $n-1$, per cui *tenendo fisso il disco* n, possiamo senz'altro spostare la pila P_{n-1} formata da $1, 2, \ldots, n-1$ dal palo A al palo B (ovviamente se esiste un modo di spostare una

pila da X a Y usando Z come appoggio esiste anche un modo per spostare la pila da X a Z usando Y come appoggio). Ció é possibile poiché, essendo n il disco piú grande, la regola degli spostamenti non viene mai violata. Una volta spostata la pila P_{n-1} su B, possiamo spostare in una mossa n da A a C e dopodiché, applicando una seconda volta l'ipotesi induttiva, spostiamo P_{n-1} da B a C. Per cui si ha

$$\begin{aligned} T(n) &= T(n-1) + 1 + T(n-1) \\ &= 2T(n-1) + 1. \end{aligned}$$

$\bigtriangledown$

Questa dimostrazione non solo dimostra l'esistenza di un metodo per spostare la torre, ma si traduce immediatamente in un algoritmo ricorsivo! Denotiamo con

```
hanoi(n, partenza, appoggio, arrivo)
```

una procedura che sposta una torre di n dischi T_n dal palo di `partenza` a quello `arrivo` usando il palo intermedio come `appoggio`. E denotiamo con con

```
muoviDisco(partenza, arrivo)
```

la semplice procedura che sposta il disco situato in cima al palo di partenza a quello di arrivo. Volendo esprimere l'algoritmo in pseudocodice, otteniamo questo:

```
procedure hanoi (n: integer; partenza, appoggio, arrivo:  palo);
begin
if (n = 1) then muoviDisco(partenza, arrivo)
else begin
   hanoi(n-1, partenza, arrivo, appoggio);
   muoviDisco(partenza, arrivo);
   hanoi(n-1, appoggio, partenza, arrivo);
end
```

2 Calma! Non c'é pericolo!

Veniamo adesso alla questione della fine del mondo. Ovvero, convinciamoci che anche se la leggenda fosse veritiera non ci sarebbe di cui preoccuparsi. Abbiamo visto che il numero delle mosse necessarie é dato dalla **equazione di ricorrenza**

$$T(n) = 2\ T(n-1) + 1$$

con condizione iniziale $T(1) = 1$. Ma quanto vale $T(n)$? Risolvere questa equazione é semplicissimo, basta non lasciarsi prendere dal panico che solitamente attanaglia gli studenti quando viene evocato lo spettro della Matematica. Il metodo consiste nell'applicare ripetutamente lo "stampo"

$$T(x) = 2\ T(x-1) + 1$$

fino a quando non risulta chiaro cosa sta succedendo e si puó indovinare la soluzione. Applichiamo quindi lo stampo ripetutamente per $x = n-1, n-2, n-3, \ldots$ ottenendo

$$\begin{aligned} T(n) &= 2\,T(n-1)+1 \\ &= 2\,(2\,T(n-2)+1)+1 \\ &= 2^2\,T(n-2)+2+1 \\ &= 2^2\,(2\,T(n-3)+1)+2+1 \\ &= 2^3\,T(n-3)+2^2+2+1 \\ &= \ldots \\ &= 2^k\,T(n-k)+2^{k-1}+\ldots+2^2+2+1 \\ &= 2^k\,T(n-k)+2^k-1. \end{aligned}$$

La somma

$$2^{k-1}+\ldots+2^2+2+1$$

é un caso particolare della **serie geometrica**

$$\sum_{i=0}^{m} q^i = \frac{q^{m+1}-1}{q-1}$$

per cui, ponendo $q=2$ si ha $2^{k-1}+\ldots+2^2+2+1 = 2^k-1$. Per $k=n-1$ si ha, ricordando che $T(1)=1$

$$\begin{aligned} T(n) &= 2^k\,T(n-k)+2^k-1 \\ &= 2^{n-1}T(1)+2^{n-1}-1 \\ &= 2^n-1. \end{aligned}$$

Se $n=64$ possiamo dormire sonni tranquilli...

Esercizio 1 *Calcolare il numero di millenni che sono necessari per spostare tutti e 64 i dischi.*

3 Implementazione C

Passiamo adesso dall'algoritmo in pseudocodice ad una implementazione C. Per prima cosa operiamo alcune scelte sulle strutture dati. Un modo semplice di rappresentare i 3 pali é di utilizzare un solo array `Poles[3 * MAXDISKS]` concettualmente diviso in 3 segmenti di `MAXDISKS` posizioni ciascuno e rappresentanti i 3 pali. Per convenzione le basi dei tre pali sono le posizioni iniziali, vale a dire 0, `MAXDISKS` e $2 \times$ `MAXDISKS`.

I tre pali sono delle *pile*, nel senso che l'ultimo disco che viene messo su un palo sará necessariamente il primo ad uscire. Un modo semplice di simulare una pila é quello di utilizzare un cursore che viene incrementato (decrementato) di 1 ogni qualvolta si inserisce (estrae) un elemento.

Con questa convenzione una semplice implementazione potrebbe essere quella di Figura 1.

Sebbene concettualmente corretta, l'implementazione di Figura 1 é errata! Il motivo é che in C i parametri sono *sempre passati per valore* e pertanto, se si vuole alterare i valori di `a, b, c` bisogna operare tramite puntatori ad essi. La versione corretta, scritta piú concisamente, compare in Figura 2

```
int Poles[MAXDISKS*3];
int a = NUM_OF_DISKS, b = MAXDISKS, c = MAXDISKS * 2;
   //pointers to 1st free position in stack of pole A, B, C
void moveDisk(int x, int y){
   x = x - 1;
   Poles[y] = Poles[x];
   y = y + 1;
   Poles[x]=0;
}
void hanoi(int n, int a, int b, int c) {
   if (n==1)
      moveDisk(a, c);
   else {
      hanoi(n-1, a, c, b);
      moveDisk(a, c);
      hanoi(n-1, b, a, c);
   }
}
```

Figure 1: Implementazione errata!

```
int Poles[MAXDISKS*3];
int a = NUM_OF_DISKS, b = MAXDISKS, c = MAXDISKS * 2;
   //pointers to 1st free position in stack of pole A, B, C
void moveDisk(int *px, int *py){
   Poles[(*py)++] = Poles[--(*px)];
   Poles[*(px)]=0;
}
void hanoi(int n, int *pa, int *pb, int *pc) {
   if (n==1)
      moveDisk(pa, pc);
   else {
      hanoi(n-1, pa, pc, pb);
      moveDisk(pa, pc);
      hanoi(n-1, pb, pa, pc);
   }
}
```

Figure 2: Implementazione corretta.

Il nome `pa` sta per *"puntatore ad* `a`" e similmente per `pb, pc` ecc. Adottando questa convenzione il codice risulta di piú facile lettura. Si noti che se `pa` é un puntatore ad `a` allora `*pa` é la cosa puntata da `pa`, cioé `a` stesso. Questo tipo di meccanismo, la gestione dei puntatori, é il cuore stesso del C e se da un lato lo rende un po' ostico, dall'altro lo rende estremamente versatile e potente. É quindi essenziale familiarizzarsi con questi meccanismi. Per far funzionare il tutto, nel `main()` si dovrá poi invocare `hanoi` in questo modo:

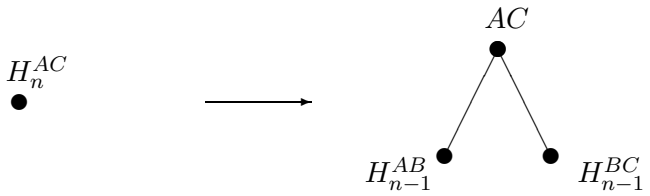
```
hanoi(NUM_OF_DISKS, &a, &b, &c)
```

avendo cioé l'accortezza di passare gli indirizzi di `a, b, c`.

Esercizio 2 *Scrivere una funzione* `swap` *che scambia i valori di due variabili intere. E.g. se* `x=1` *e* `y=3` *dopo* `swap` *si deve avere* `x=3` *e* `y=1`*.*

4 Attraversamento di alberi e generazione delle mosse

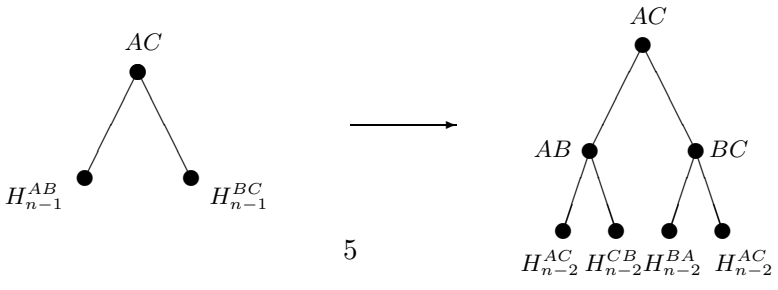
Il nostro scopo adesso é quello di riscrivere il programma per le Torri di Hanoi in modo completamente iterativo. Ció avverrá analizzando il comportamento del programma ricorsivo. Questi genererá una sequenza di chiamate ricorsive le quali, prima o poi, si tradurranno in una sequenza di mosse per effettuare lo spostamento. Iniziamo col vedere la sequenza delle chiamate ricorsive. Per comoditá denoteremo una invocazione di `hanoi(n, X, Y, Z)` come H_n^{XY} mentre per `muoviDisco(X,Y)` scriveremo solamente XY. Il comportamento di `hanoi(n, X, Y, Z)` (cioé H_n^{XY}) puó essere schematizzato come segue:



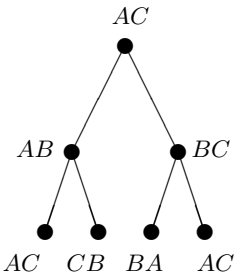
Il grafico significa semplicemente che la procedura `hanoi(n, X, Y, Z)` chiamerá, in sequenza,

1. `hanoi(n-1, X, Z, Y)`,
2. `hanoi(1, X, Y, Z)`, cioé `muoviDisco(X,Y)`, e
3. `hanoi(n, Z, Y, X)`.

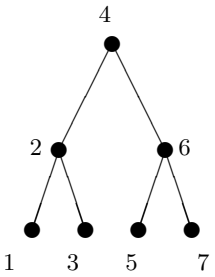
A sua volta H_{n-1}^{XZ} genererá le chiamate H_{n-2}^{XY}, H_1^{XZ} e H_{n-2}^{YZ}, mentre H_{n-1}^{ZY} genererá le chiamate H_{n-2}^{ZX}, H_1^{ZY} e H_{n-2}^{XY}:



Questo processo di sostituzioni successive termina perché ogni volta che si sostituisce H_n^{XY} si ottengono 3 chiamate ricorsive con parametro $n-1$, 1 ed $n-1$, per cui prima o poi si ottiene un albero con esattamente n livelli i cui nodi sono etichettati con mosse (chiamate a muovidisco). Ad esempio, se sviluppassimo per intero l'albero delle chiamate generato da `Hanoi(3, A, C, B)` otterremmo:



L'albero contiene tutte le mosse, ma in quale ordine devono essere eseguite? La risposta é contenuta nel seguente diagramma:



Questa etichettatura si ottiene quando l'albero binario viene attraversato secondo il metodo **in-order**. L'etichettatura segue fedelmente l'ordine con cui sono eseguite le chiamate ricorsive, vale a dire, quando `Hanoi(n, X, Y, Z)` viene eseguita:

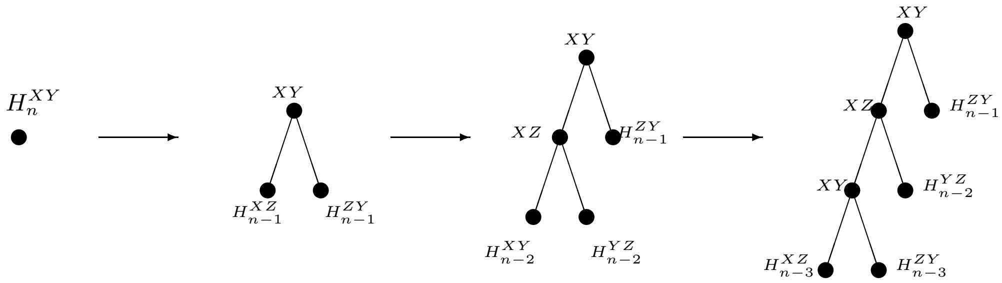
- primo, viene invocata ed eseguita `Hanoi(n-1, X, Z, Y)`;
- secondo, viene invocata ed eseguita `Hanoi(1, X, Y, Z)`;
- terzo, viene invocata ed eseguita `Hanoi(n-1, Z, Y, X)`.

Dato che a sua volta `Hanoi(n-1, X, Z, Y)` genera tre chiamate ricorsive, sia `Hanoi(1, X, Y, Z)` che `Hanoi(n-1, Z, Y, X)` prima di essere eseguite dovranno aspettare che vengano concluse le chiamate ricorsive generate da `Hanoi(n-1, X, Z, Y)`, le quali a loro volta genereranno altre chiamate ricorsive, le quali genereranno altre chiamate ricorsive, e cosí via.

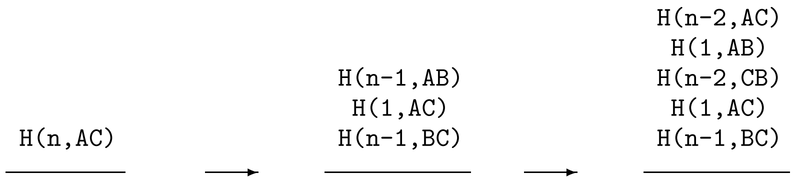
Esercizio 3 *Scrivere un programma ricorsivo che etichetti un albero binario secondo il metodo in-order.*

5 Eliminazione della ricorsione

Se dovessimo tracciare la sequenza in cui le chiamate ricorsive vengono “generate” otterremmo una sequenza del tipo:



...dove per semplicitá tipografica XY rappresenta H_1^{XY}. Se dovessimo simulare “a mano”, su un foglio di carta, l’algoritmo ricorsivo, per non commettere errori sarebbe opportuno annotare di volta in volta le chiamate ricorsive man mano generate ma non ancora completate. Otterremmo una sequenza del tipo:



```
hanoiIterativo(N, X, Y);

begin
  push(n, X, Y);
  repeat
    (n,X,Y) := pop();
    if (n=1) then muoviDisco(X,Y);
    else begin
      push(n-1,ZY);
      push(1,XY);
      push(n-1,XZ);
      end;
  until stackEmpty();
end
```

Figure 3: versione iterativa delle Torri di Hanoi

In altre parole, l'algoritmo é il seguente:

- inizializza il processo mettendo `H(N,XY)` sullo stack;
- Esegui una pop. Sia `(n, XY)` l'elemento restituito dalla pop.
  - se $n = 1$ allora esegui `muovidisco(XY)`;
  - se $n > 1$ allora esegui, nell'ordine, `push(n-1,ZY)`, `push(1,XY)` e `push(n-1,XZ)`;
  - se la pop segnala che la pila é vuota, stop!

L'algoritmo risultante appare in pseudocodice in Figura 3.

Esercizio 4 *Si simuli a mano* `hanoiIterativo(3, A, C)`.

Esercizio 5 *Si costruisca l'albero delle mosse generato da* `hanoi(2, A, C, B)` *e lo si etichetti in modo da mostrare l'ordine di esecuzione delle mosse.*