Programmazione II
Università di Roma "La Sapienza"
Appunti a cura della Prof.ssa FACHINI

Ricorsione per il "problem solving"

Il problema del cambio.

Consideriamo il problema di determinare in quanti modi diversi si può cambiare una somma di denaro. Per esempio potremmo voler sapere quanti modi diversi ci sono di cambiare 100 euro. In tal caso avremmo a disposizione biglietti da 5,10,20,50,100 euro, monete da 1 euro, da 50,20,10,5,2 centesimi e infine da 1 centesimo.
Il problema consiste nel progettare una procedura per calcolare il numero di modi di cambiare una data quantità di danaro, avendo a disposizione *n* tagli di monete.
Vediamo un caso "piccolo": in quanti modi possiamo cambiare 10 centesimi, avendo a disposizione monete da 10,5, 2 e 1 centesimo.
Mettiamo in Ncambi il risultato, mano mano che viene calcolato.
Innanzi tutto ho la moneta da10 centesimi, quindi Ncambi = 1.
Ma possiamo usare monete da 5: possiamo usarla due volte, una o non usarla affatto. Nel primo caso otteniamo un nuovo cambio, Ncambi = 2.
Nel secondo si tratta di vedere in quanti modi posso cambiare la cifra rimanente, 5 centesimi. Nel terzo dobbiamo considerare tagli più piccoli, per l'intera cifra. Consideriamo il secondo caso: possiamo usare monete da 2 e a 1 per cambiare i 5 centesimi.
Potremmo usare 1 volta o 2 volte la moneta da 2 e vedere in quanti modi diversi possiamo cambiare rispettivamente 3 centesimi o 1 centesimo, oppure non usarla affatto.
Nel secondo caso possiamo usare solo una moneta da 1, e quind Ncambi = 3.
Se non usiamo le monete da 2, possiamo solo usare 5 monete da 1, quindi Ncambi = 4
Nel primo caso notiamo che se cambiamo i 3 centesimi con una moneta da 2, allora resta un solo centesimo, Ncambi = 5 oppure cambiamo i 3 centesimi con 3 da 1, quindi Ncambi = 6.
Resta da vedere in quanti modi cambiare i 10 centesimi senza usare le monete da 5, poiché il taglio rimanente è solo 1, questo numero è pari al numero di volte in cui si può usare la moneta da 2 che è 5, da cui Ncambi = 11.

Esaminando la costruzione della soluzione possiamo definire un algoritmo ricorsivo a questo problema.
Supponiamo di ordinare i tagli di monete disponibili (se l'ordine è decrescente o crescente, non fa differenza). Osserviamo che i modi di cambiare una somma possono essere divisi in due gruppi: quelli che utilizzano il primo taglio e quelli che non lo fanno. Quindi il numero totale di modi di cambiare una somma di denaro può essere calcolato sommando il numero di modi di cambiarla senza usare il primo taglio, più il numero di modi di cambiarla assumendo che quel taglio venga usato almeno una volta. Ma quest'ultimo numero è uguale al numero di modi di cambiare la somma di partenza diminuita del valore del taglio, usando di nuovo tutti i tagli a disposizione.
Allora possiamo dire che il numero di modi di cambiare una quantità di denaro ***somma*** usando ***n*** tagli è uguale al

> numero di modi di cambiare ***somma*** usando tutti tranne il primo
> più
> il numero di modi di cambiare ***somma-t1,*** dove ***t1*** è il valore del primo taglio, usando tutti gli ***n*** tagli.

Così si riduce il calcolo ricorsivamente a somme più piccole e a minor numero di tagli.
Per poter tradurre in programma questa soluzione dobbiamo risolvere i casi base:

Se somma è 0, il numero di modi di cambiarla è 1
Se somma è <0 allora il numero di modi di cambiarla è 0
Se n è 0 allora ci sono 0 modi di cambiare la somma data!
Questa soluzione si traduce facilmente nella seguente funzione ricorsiva:

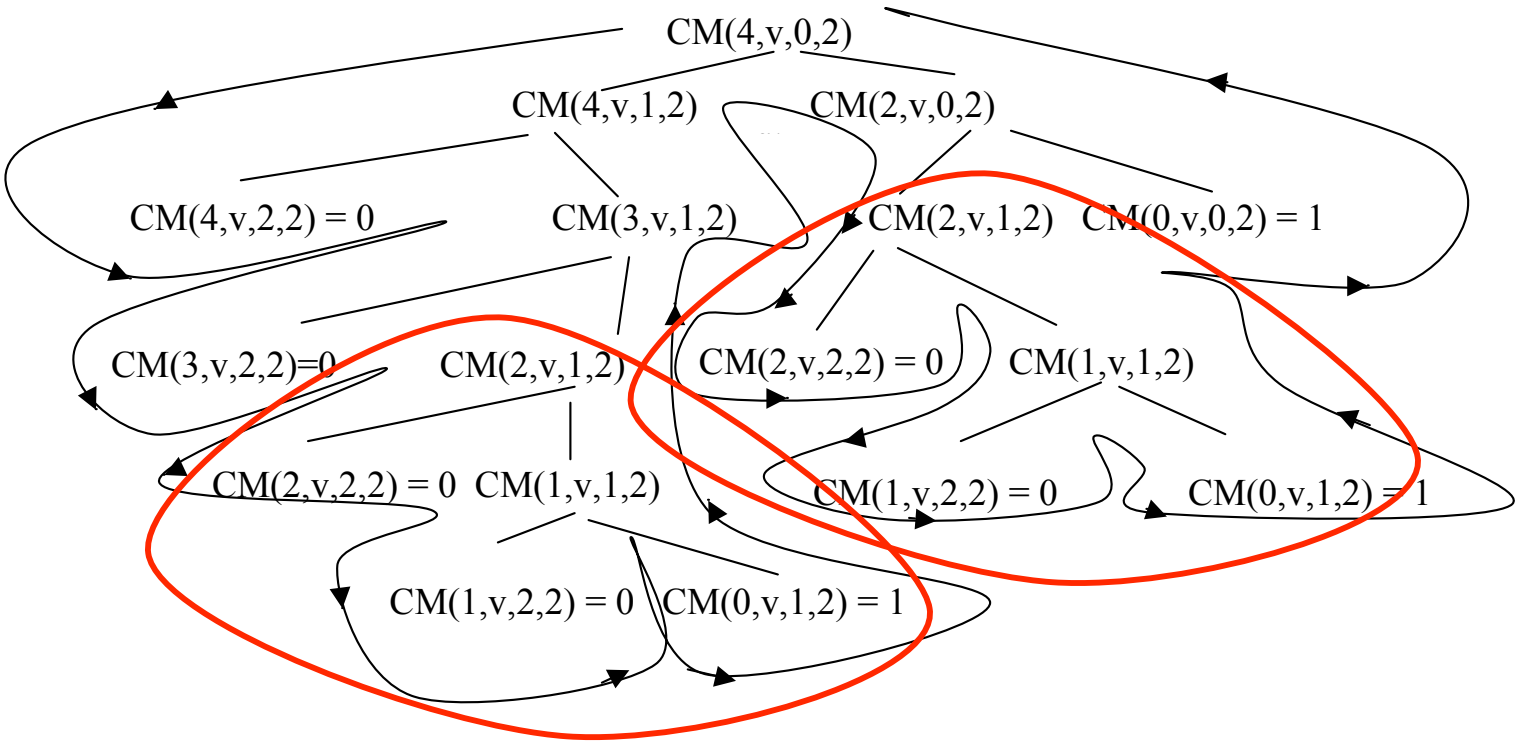
Questa funzione deve essere chiamata con `i` = 0 e `n` pari al numero dei tagli disponibili.

```
int cambiaMonete(float somma,float* tagli, int i,int n)
{if (somma < 0.0 || n == i) return 0;
if (somma == 0.0) return 1;
return cambiaMonete(somma,tagli,i+1, n) +
```

```
        cambiaMonete(somma-tagli[i],tagli,i,n);
}
```

Questa funzione genera un albero delle chiamate con ridondanze analoghe a quello che abbiamo visto per il calcolo dei numeri di Fibonacci. Ma non è così immediato costruirne una equivalente più efficiente.

In questo esempio abbiamo preso solo due tagli nel vettore v=[2,1]



D'altra parte non è affatto ovvio come progettare un algoritmo migliore per calcolare il risultato.

Le torri di Hanoi.

Anche il classico problema della Torre di Hanoi è un esempio di uso della ricorsione nel proble solving:

Nella sua forma classica, quella che si trova in molti negozi di giochi, la Torre di Hanoi è formata da otto dischi sovrapposti, di dimensioni decrescenti, bucati al centro e infilati in una delle tre colonnine fissate su una tavoletta.

Gli otto dischi, che formano la cosiddetta *torre*, devono poi essere spostati su una delle altre due colonnine libere, seguendo però una regola precisa: si può spostare soltanto un disco alla volta ed è proibito collocare uno qualsiasi dei dischi su uno più piccolo.

Il gioco venne inventato nell'Ottocento da Edouard Lucas, studioso di teoria dei numeri, che deve la sua fama all'analisi della successione di Fibonacci. Professore di matematica in un liceo parigino, Lucas è uno dei più grandi inventore di giochi matematici, di tanti rompicapi e giochi ancora oggi popolari. Lucas lanciò il suo nuovo gioco nel 1883 e quella che segue, è la sua presentazione originale, dal tono ottocentesco, un po' ingenuo, ma divertente:

Questo gioco è stato scoperto, per la prima volta, fra le carte dell'illustre Mandarino FER-FER-TAM-TAM, e sarà pubblicato nell'immediato futuro, su ordine del governo cinese.

La ***Torre di Hanoi*** *è composta da una serie dischi, di dimensioni decrescenti, in numero variabile, che noi abbiamo realizzato con otto dischi di legno, bucati al centro. In Giappone, in Cina e a Tonkino sono di porcellana.*

Il gioco consiste nel demolire la torre e nel ricostruirla su un'altra

colonnina, seguendo le regole date.
Divertente e istruttivo, facile da imparare e da giocare in città, in campagna oppure in viaggio, ha per obiettivo la divulgazione della scienza, come tutti gli altri giochi originali e curiosi del professor ***N. CLAUS (OF SIAM)***

Indichiamo con A, B e C le tre colonnine. Nel caso banale di un unico disco, occorrerà un solo movimento per risolvere il gioco, basta infatti spostare il disco dalla colonnina A alla colonnina C. Con due dischi, sono necessari 3 movimenti, si deve spostare il disco superiore sulla colonnina B, il disco più grande su C ed infine l'altro disco sempre su C. Quali sono gli spostamenti minimi necessari per trasferire la torre a tre dischi da A a C? Dobbiamo spostare il disco superiore su C e quello di mezzo su B, sul quale spostiamo poi il disco più piccolo.
In seguito, spostiamo il disco più grande su C, quello più piccolo su A e, per finire, il disco da B a C e da A a C. Sono, in totale, sette movimenti.
Con un po' di pratica si arriva facilmente a capire il procedimento da seguire con un numero qualsiasi di dischi, scoprendo la formula risolutiva del gioco. Se lo sappiamo risolvere con tre dischi, lo sapremo, infatti, risolvere anche con quattro. Sarà sufficiente trasportare dapprima i tre dischi superiori sulla seconda colonnina, con il procedimento già noto, successivamente il quarto disco sulla terza e infine si collocheranno su questo gli altri tre dischi, sempre con procedimento già utilizzato in precedenza. Esprimiamo la rgola di soluzione ricorsivamente:

- Dato *n* il numero dei dischi
- Si numerano i dischi da 1 (il più piccolo, in alto) a *n* (il più grande, in basso)

Per spostare i dischi dal paletto A al paletto B:
1 Sposta i primi *n* -1 dischi da A a C, usando il paletto B come appoggio.
Questo lascia il disco *n* da solo sul paletto A,

2 Sposta il disco *n* da A a B,

3 Sposta *n* -1 dischi da C a B, usando il paletto A come appoggio

Questa strategia si traduce facilmente in una funzione C:

```
void hanoi(int n, int part, int arr, int aus)
```

```
{if (n > 0)
     {hanoi(n-1,part,aus,arr);
     /* Stampa a video */
     printf("Da %d a %d\n",part,arr);
     hanoi(n-1,aus,arr,part);
     }
}
```

Se si esamina l'albero delle chiamate di hanoi, si vede subito che si tratta di un albero con 2^{n-1} foglie. D'altra parte non c'è modo di costruire una soluzione più efficiente.