Olimpiadi di Informatica 2009
Giornate preparatorie

Dipartimento di Informatica
Università di Torino

Elio Giovannetti

marzo 2009

Lino il giornalaio.

Caratterizzazione ricorsiva

Siano val[0], val[1], ..., val[n-1] i rispettivi valori degli n tipi di monete a disposizione.

Il numero di modi nModi(S, i) in cui si può realizzare la somma S con monete di valori da val[0] a val[i] inclusi (non necessariamente usando tutti i valori da val[0] a val[i]) è dato da:

il numero di modi in cui si può realizzare la somma S con monete di valori da val[0] a val[i-1], cioè senza usare monete di valore val[i]

+

il numero di di modi in cui si può realizzare la somma S con monete di valori da val[0] a val[i] che usano almeno una moneta di valore val[i].

Ma si ha:

- il numero di modi in cui si può realizzare la somma S con monete di valori da val[0] a val[i-1] è nModi(S, i-1) ;
- il numero di di modi in cui si può realizzare la somma S con monete di valori da val[0] a val[i] che usano almeno una moneta di valore val[i] si ottiene nel modo seguente:
  - si sottrae dalla somma S una moneta val[i], e poi si conta in quanti modi si può ottenere tale somma S – val[i] usando monete di valori da val[0] a val[i];

    quindi è nModi(S – val[i], i) .

    Nota bene: se in tale conto usassimo solo monete di valori da val[0] a val[i-1] otterremmo solo il numero di modi di realizzare S usando una sola moneta val[i].

Quindi: nModi(S, i) = nModi(S, i-1) + nModi(S – val[i], i) ;

Base della ricorsione

- La somma 0 può essere realizzata in un modo solo, quali che siano le monete permesse (ed è formata da 0 monete): nModi(0, i) = 1 per qualunque i.
- Una somma S diversa da 0 non può essere realizzata senza monete, quindi:

  nModi(S, -1) = 0 per qualunque S ≠ 0

  (nota che "senza monete" vuol dire i = -1, perché per i = 0 si ha il primo tipo di moneta, di valore val[0]); più in generale potremmo scrivere nModi(S, i) = 0 per i < 0 (ed S ≠ 0).
- Una somma S negativa non può essere realizzata, quali che siano le monete permesse; quindi:

  nModi(S, i) = 0 per qualunque i, se S < 0. Nota che questo caso base si verifica quando si sottrae una moneta di valore maggiore della somma rimasta.

La caratterizzazione ricorsiva può essere tradotta direttamente in una funzione ricorsiva C o Pascal

```
long nModi(int s, int i) {
  if(s == 0) return 1;
  if(s < 0) return 0;
  if(i < 0) return 0;
  return nModi(s, i-1) + nModi(s-val[i], i);
}
```

Nota Bene: è essenziale che il test s == 0 venga fatto prima del test i < 0, affinché per s == 0 la funzione restituisca 1 anche se i < 0.

Versione iterativa

La soluzione ricorsiva precedente è inefficiente, perché ricalcola molte volte gli stessi risultati parziali.

Una soluzione più efficiente si ottiene con una realizzazione iterativa che memorizza in un array i risultati parziali via via ottenuti, che possono così essere riutilizzati senza bisogno di ricalcolarli (tecnica della *programmazione dinamica*).

Versione iterativa

Siano N il numero dei differenti valori di monete,
R la somma da realizzare con le monete.

Si tiene un array num indiciato da 0 fino al valore R, e si effettuano N successive scansioni dell'array:
al termine dell'i-esima scansione ogni elementi num[s] deve contenere il num. di modi in cui si può realizzare la somma parziale s con monete di valori da val[0] a val[i]. Al termine delle N scansioni, nell'ultimo elemento dell'array ci sarà il risultato. Lo schema è quindi:

```
for (int i = 0; i < N; i++) {
   ...
   for (int s = ...; s <= R; s++)
        ...
}
```

All'inizio dell'i-esima iterazione del ciclo esterno,

num[s] contiene il n. di modi in cui si può realizzare la somma s con monete da val[0] a val[i-1] (incluse).

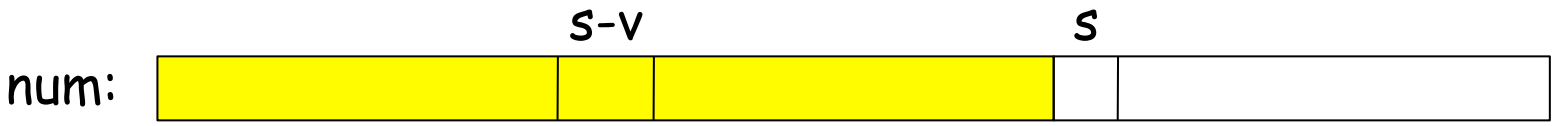
Vogliamo aggiornare ciascun num[s] in modo che contenga il n. di modi della somma s con monete fino a val[i] (inclusa).

Notiamo che per s < val[i] non si possono usare monete di valore val[i], e quindi il n. di modi num[s] rimane lo stesso.

I due for annidati devono avere quindi la forma:

```
for (int i = 0; i < N; i++) {
   int valore = val[i];
   for (int s = valore; s <= R; s++)
      ...
}
```

Situazione al passo generico, con indici i ed s.



Per $s' < s$ l'array è già stato aggiornato, quindi è

`num[s']` = nModi(s', i);

cioè `num[s-v]` = nModi(s-v, i);

Per $s' \geq s$ l'array deve ancora essere aggiornato, quindi è

`num[s']` = nModi(s', i-1);

e in particolare è `num[s]` = nModi(s, i-1);

Vogliamo fare in modo che `num[s]` diventi nModi(s, i); ma sappiamo che è

nModi(s, i) = nModi(s, i-1) + nModi(s - val[i], i) ;

quindi basterà fare:

```
num[s] = num[s] + num[s-val[i]];
```

Inizializzazione

La somma 0 può essere realizzata in un solo modo, quindi porremo inizialmente num[0] = 1, e tutti gli altri elementi uguali a zero:

```
num[0] = 1;
for(int s = 1; s <= R; s++) num[s] = 0;
```

Nota: in C++ se l'array num è dichiarato globalmente, cioè fuori dal main, esso viene inizializzato automaticamente a 0, e quindi il ciclo for scritto qui sopra non è necessario.

Conclusione

```
int main() {
  fin >> N >> R;
  for(int i = 0; i < N; i++)
    fin >> val[i];  // fine parte di input

  num[0] = 1;
  for(int s = 1; s <= R; s++) num[s] = 0;
  for (int i = 0; i < N; i++) {
    int valore = val[i];
    for (int s = valore; s <= R; s++)
      num[s] += num[s-valore];
  }

  cout << num[R] << endl;
  fout << num[R];
}
```