# Linguaggi di Programmazione: progetto "The Change Making Problem"

A.A. 2012-2013

## Il problema

Il Change Making proglem (CMP) è un classico problema di ottimizzazione combinatoria. E' dato un insieme $I$ di tipi di monete ed una somma da raggiungere; ad ogni tipo di moneta $i \in I$ è associato il valore $w_i$ ed un valore $p_i$; la somma da raggiungere è indicata con $C$. Supponiamo che i dati del problema siano interi non negativi. L'obiettivo è scegliere un sottoinsieme di monete $\bar{I} \subseteq I$, ed un numero di monete $x_i$ per ciascun tipo scelto, in modo che il valore totale delle monete scelte sia pari a $C$

$$\sum_{i \in \bar{I}} w_i \cdot x_i = C$$

ed il numero totale di monete scelte

$$\sum_{i \in \bar{I}} x_i$$

sia minimo. Non sono ammessi frazionamenti di monete. Il CMP è una variante del Knapsack Problem (KP): richiede di ottimizzare una funzione obiettivo lineare e la soluzione è soggetta ad un singolo vincolo, esprimibile come una equazione lineare. Le condizioni di integralità sulla scelta degli oggetti, tuttavia, lo rendono un problema NP-Hard [1]: non sono noti algoritmi risolutivi con costo in tempo polinomiale.

# Un algoritmo

In letteratura sono stati proposti diversi algoritmi per ottenere delle soluzioni esatte [2]. Un famoso algoritmo risolutivo, che sfrutta tecniche di programmazione dinamica [3], è il seguente.

Senza perdita di generalità, supponiamo che i tipi di monete nell'insieme $I$ siano ordinati secondo un criterio arbitrario, indichiamo con $n = |I|$ il numero di tipi di monete, e supponiamo quindi l'insieme $I$ possa essere rappresentato come $I = \{1, 2, \ldots, n\}$. Indichiamo con $f_{kc}$ il *valore* della *miglior combinazione di monete* che

(a) include solo tipi di monete di indice compreso tra 1 e $k$;

(b) ha somma dei valori pari a $c$.

Il valore

$$\{f_{nC}\}$$

corrisponde alla soluzione ottima del problema (può includere qualsiasi tipo di moneta ed ammette una somma dei valori pari a $C$).

Intuitivamente, la miglior soluzione ad un problema che include solo tipi di monete di indice compreso tra 1 e $k$ ed utilizza $c$ unità di capacità può essere ottenuta analizzando due soli casi:

- **non includendo** il tipo $k$-esimo, ed in tal caso la miglior soluzione è la stessa che si ottiene considerando solo tipi di indice compreso tra 1 e $k - 1$;
- **includendo** $x_i$ esemplari del tipo $k$-esimo, ed in tal caso la miglior soluzione è si ottiene utilizzando $x_i \cdot w_k$ unità di capacità per il nuovo tipo, e sfruttando nel modo migliore le rimanenti $c - x_i \cdot w_k$ unità di capacità per i tipi precedenti.

Quindi, per ogni $k \in \{0 \dots n\}$ e per i valori di capacità utilizzata $c \in \{0 \dots C\}$, i valori $f_{kc}$ possono essere calcolati in modo ricorsivo come segue:

$$f_{kc} = \begin{cases} 0 & \text{se } k = 0 \text{ e } C = 0 \\ +\infty & \text{se } k = 0 \text{ e } C > 0 \\ f_{k-1,c} & \text{se } k > 0 \text{ e } w_k > c \\ \min(1 + \min_{x_i=1\dots\lfloor c/w_k \rfloor}\{f_{k-1,c-x_i \cdot w_k}\}, f_{k-1,c}) & \text{altrimenti.} \end{cases} \tag{1}$$

o, tramite una piccola semplificazione, come segue:

$$f_{kc} = \begin{cases} 0 & \text{se } k = 0 \text{ e } C = 0 \\ +\infty & \text{se } k = 0 \text{ e } C > 0 \\ f_{k-1,c} & \text{se } k > 0 \text{ e } w_k > c \\ \min(1 + f_{k,c-w_k}, f_{k-1,c}) & \text{altrimenti.} \end{cases} \tag{2}$$

La valutazione di tale funzione ricorsiva può avvenire per via diretta, ed in tal caso il relativo costo in tempo è esponenziale, oppure sfruttando tecniche di memoization tipo accumulatori (come visto a lezione per i numeri di Fibonacci o per il calcolo del fattoriale), ed in tal caso tutti i valori $f_{kc}$ possono essere calcolati in tempo $O(nC)$ (pseudo-lineare).

# Parallelizzazione

Il calcolo dei valori $f_{kc}$ si presta bene ad un approccio concorrente.

**Concorrenza a livello di sottoprogrammi:** ogni valore di $f_{kc}$ può essere calcolato tramite l'attivazione di un task indipendente; tali task devono essere sincronizzati, permettendo l'esecuzione del task per $f_{kc}$ solo quando i task per $f_{k-1,c}$ e $f_{k,c-w_k}$ hanno terminato la propria esecuzione.

**Concorrenza a livello di statement:** fissato un valore di $k$, tutti i termini $f_{kc}$ possono essere calcolati in modo indipendente. Quindi un approccio concorrente consiste nel calcolo dei valori $f_{kc}$ "riga per riga": si parte con $k = 0$ ed in parallelo si calcolano i valori $f_{0c}$ per $0 \leq c \leq C$, si passa a $k = 1$ calcolando i valori $f_{1c}$ in parallelo e via dicendo (in questo caso non è però possibile utilizzare la forma (2)).

# Consegna

Realizzare cinque diverse implementazioni dell'algoritmo proposto, sfruttando le seguenti metodologie:

- programmazione imperativa (eventualmente strutturata ad oggetti); es. C o Java;
- programmazione funzionale; es. Scheme;
- programmazione dichiarativa; es. PROLOG;
- programmazione imperativa con concorrenza a livello di sottoprogrammi;
- programmazione imperativa con concorrenza a livello di statement.

Preparare una breve relazione, tralasciando la discussione dell'algoritmo, ma concentrandosi sul **confronto critico tra le implementazioni realizzate**: *pregi, difetti, limiti, differenze, peculiarità, impatto sul costo ecc.*

# Riferimenti bibliografici


[1] M.R. Garey, D.S. Johnson (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman.

[2] S. Martello, P. Toth (1990) Knapsack problems – algorithms and computer implementations, John Wiley and sons.

[3] A.V. Aho, J.E. Hopcroft, J.D. Ullman (1983) Data Structures and Algorithms, Addison-Wesley.