Progettazione di Algoritmi **Anno Accademico 2023–2024**

Note per la Lezione 13

Ugo Vaccaro

Ricordiamo che lo sviluppo di algoritmi basati sulla Programmazione Dinamica prevede generalmente due passi separati:

- **Formulare il problema in termini ricorsivi**: ovvero scrivere una espressione per la soluzione all'intero problema che sia una combinazione di soluzioni a sottoproblemi di taglia minore
- **Calcolare la soluzione globale al problema in modo ricorsivo,** facendo precedere ciascuna chiamata ricorsiva con un controllo per verificare se la soluzione al relativo sottoproblema è stata già calcolata.

Gli algoritmi di Programmazione Dinamica hanno bisogno di memorizzare le soluzioni ai sottoproblemi intermedi. Spesso (ma non sempre) ciò viene effettuato memorizzandole in tabelle.

Il secondo passo prima descritto può essere sostituito in algoritmi iterativi con il seguente:

- **Calcolare le soluzioni ai sottoproblemi in maniera "bottom-up"**: scrivere un algoritmo che parta con i casi base della ricorrenza e proceda via via considerando (e risolvendo) problemi di taglia sempre maggiore, considerandoli nell'ordine corretto

Applichiamo la tecnica di Programmazione Dinamica al seguente problema.

Cambio delle monete.

Input: Un valore monetario V, un insieme di monete che denoteremo con l'insieme $\{1, \ldots, n\}$, i cui valori (ad es., in Euro)) sono contenuti nel vettore $v = v[1] \ldots v[n]$, con $v[1] > v[2] > \ldots > v[n] = 1$. In altri termini, la moneta generica i vale $v[i]$ Euro.

Output: Il minimo numero di monete il cui valore totale sia esattamente pari a V. (Assumiamo di avere a disposizione un numero illimitato di monete di valore $v[i]$, per ogni i)

In altri termini, indicato con $a_i \geq 0$ il numero di monete di valore $v[i]$ che usiamo (che può anche essere pari a zero, nel senso che la moneta i-esima non viene utilizzata), vogliamo *minimizzare* il numero totale di monete usate, pari a

$$a_1 + a_2 + \ldots + a_n$$

sotto la condizione che

$$\sum_{i=1}^{n} a_i v[i] = V$$

Vediamo un esempio.

Sia $V = 26$ (Euro), l'insieme delle monete pari a $\{1, 2, 3, 4\}$ ed il vettore dei valori delle monete dato da

$$v[1] = 10, v[2] = 5, v[3] = 2, v[4] = 1$$

In altri termini, abbiamo a disposizione monete di valore pari a 10 Euro, monete di 5 Euro, monete di 2 Euro e monete del valore di 1 Euro. Vogliamo "cambiare un assegno" di 26 Euro, usando il minor numero di monete possibili.

Vi sono diverse possibili soluzioni per esprimere 26 Euro con le monete a disposizione:

1. $a_1 = a_2 = a_3 = 0$, $a_4 = 26$ $\Rightarrow$ il numero totale di monete usato è $a_1 + a_2 + a_3 + a_4 = 26$, di valore totale $\sum_{i=1}^{4} a_i v[i] = 26$

2. $a_1 = 2$, $a_2 = 0$ $a_3 = 3$, $a_4 = 0$ $\Rightarrow$ il numero totale di monete usato è $a_1 + a_2 + a_3 + a_4 = 5$ di valore totale $\sum_{i=1}^{4} a_i v[i] = 2 \cdot 10 + 3 \cdot 2 = 26$

3. $a_1 = 2$ $a_2 = 1$ $a_3 = 0$, $a_4 = 1$ $\Rightarrow$ il numero totale di monete usato è $a_1 + a_2 + a_3 + a_4 = 4$ di valore totale $\sum_{i=1}^{4} a_i v[i] = 2 \cdot 10 + 1 \cdot 5 + 1 \cdot 1 = 26$

Passo 1 di PD: Formulare il problema ricorsivamente:

Ovvero: scrivere una formula per la soluzione all'intero problema che sia una combinazione di soluzioni a sottoproblemi di taglia minore

- Domanda: E quali sono i sottoproblemi del problema di partenza (che chiede di esprimere il valore V usando il minor numero di monete, ognuna delle quali di un possibile valore $v[1] > v[2] > \ldots > v[n] = 1$)?
- Risposta: Sono tutti i sottoproblemi che si ottengono qualora si voglia esprimere un qualsiasi valore $0 \leq j \leq V$ usando il minor numero di monete, ognuna delle quali di un possibile valore $v[i] > v[i+1] > \ldots > v[n] = 1$, $i \geq 1$

In altri termini, un generico sottoproblema è individuato dal generico valore j, per $j = 0, \ldots, V$ e dal sottoinsieme di monete $\{i, \ldots, n\}$, ovvero dal sottovettore dei loro valori $v[i] \ldots v[n]$.

Denotiamo con $C(i, j)$ il minimo numero di monete necessario per esprimere la somma $j \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$, (noi siamo interessati a $C(1, V)$)

Consideriamo un esempio con $V = 12$, e monete di valore $v[1] = 10$, $v[2] = 6$, $v[3] = 1$. Nella matrice di seguito riportata l'indice di riga i specifica che sono disponibili le monete di valore $v[i], \ldots, v[3]$. L'indice di colonna j specifica il valore monetario totale che si deve esprimere. La generica entrata nella riga i e colonna j della matrice indica il minor numero di monete necessario per poter esprimere il valore monetario j, $0 \leq j \leq 12$ con le monete di valore $v[i], \ldots, v[3]$, con $1 \leq i \leq 3$.

								j						
		0	1	2	3	4	5	6	7	8	9	10	11	12
	1	0	1	2	3	4	5	1	2	3	4	1	2	2
i	2	0	1	2	3	4	5	1	2	3	4	5	6	2
	3	0	1	2	3	4	5	6	7	8	9	10	11	12

Ad esempio, $C(2, 8) = 3$, dovendo necessariamente usare una moneta di valore 6 e due monete di valore 1.

Vediamo ora come esprimere in maniera ricorsiva i valori $C(i, j)$.

Ricordiamo che $C(i, j)$ è il *minimo* numero di monete necessario per esprimere la somma $j \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$. Possono accadere due casi:

- Nella soluzione che ci fornisce il minimo numero di monete $C(i, j)$ per esprimere la somma $j \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$, di fatto *non compare* la moneta i-esima di valore $v[i]$. Ne segue che $C(i, j)$ può essere calcolata tenendo in cosniderazione solo le monete di valore $v[i+1] > \ldots > v[n]$. Detto in altri termini, in questo caso vale che $C(i, j) = C(i+1, j)$.

- Può invece accadere che nella soluzione che ci dà il minimo numero di monete $C(i, j)$ per esprimere la somma $j \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$, la moneta i-esima di valore $v[i]$ *appare*. In questo caso, vuol dire che $C(i, j) = 1 + k$, dove l'1 conta l'apparenza della moneta di valore $v[i]$, e il k conta le monete restanti usate, siano esse $i_1, \ldots, i_k$ (ricordiamo che alcune di queste possono anche essere uguali). I valori di tali k monete restanti devono necessariamente sommare a $j - v[i]$ (visto che poi sommando la moneta di valore $v[i]$, che sappiamo esserci, arriviamo al valore totale j). L'osservazione chiave è che che k non è un numero sconosciuto, ma k è proprio uguale a $C(i, j - v[i])$. Detto in altre parole, k è il *minimo* numero di monete per esprimere la somma $j - v[i] \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$. Infatti, se k fosse $> C(i, j - v[i])$, allora potremmo sostituire le monete $i_1, \ldots, i_k$ con le $C(i, j - v[i])$ monete che sò, per definizione di $C(\cdot, \cdot)$, avere come somma di valori uguale a $j - v[i]$, ed ottenere, usando poi la moneta di valore $v[i]$, un valore totale pari a $(j - v[i]) + v[i] = j$, avendo impiegato un numero di monete pari a $1 + C(i, j - v[i]) < 1 + k = C(i, j)$, contro l'ipotesi che $C(i, j)$ è il *minimo* numero di monete per esprimere la somma $j \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$. Se invece k fosse $< C(i, j - v[i])$, l'assurdo sarebbe ancora più immediato. Infatti, avremmo che con le monete $i_1, \ldots, i_k$ (che sono in numero inferiore a $C(i, j - v[i])$) riusciremmo ad esprimere il valore $j - v[i]$, contro l'ipotesi che $C(i, j - v[i])$ è il *minimo* numero di monete per esprimere la somma $j - v[i] \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$. Riassumendo, abbiamo dimostrato che nel caso in cui nella soluzione che ci dà il minimo numero di monete per esprimere la somma $j \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$, la moneta i-esima di valore $v[i]$ *appare*, allora vale che $C(i, j) = 1 + C(i, j - v[i])$.

Poichè noi a priori *non sappiamo* se nella soluzione che ci dà il minimo numero di monete $C(i, j)$ per esprimere la somma $j \leq V$, usando monete di valore $v[i] > v[i+1] > \ldots > v[n]$, la moneta i-esima di valore $v[i]$ *appare* o *non appare*, ci calcoliamo le sottosoluzioni $C(i+1, j)$ e $1 + C(i, j - v[i])$ ad *entrambi* i sottoproblemi e ci prendiamo la migliore (ovvero la migliore soluzione tra quella che contiene la moneta di valore $v[i]$ e quella che non contiene $v[i]$). Detto in altri termini, vale che

$$C(i, j) = \begin{cases} C(i+1, j) & \text{se } v[i] > j, \\ \min\{C(i+1, j), 1 + C(i, j - v[i])\} & \text{se } v[i] \leq j \end{cases}$$

con i casi base della ricorrenza pari a:

$$C(n, j) = j, \quad \forall j = 0, \ldots, V.$$

Il secondo passo nella applicazione della tecnica Programmazione Dinamica consisterà in:

Passo 2 di PD: Introduci una tabella dove memorizzare il computo delle sottosoluzioni al problema di partenza:

Ovvero: calcola la soluzione ai distinti sottoproblemi una volta soltanto, memorizza ciascuna sottosoluzione nella entrata opportuna di una tabella $T[i, j]$, in modo tale che esse possano essere usate nel seguito, se occorre.

Abbiamo quindi il seguente algoritmo ricorsivo per il calcolo di $C(1, V)$:

```
Rec_CambioMonete(v[i]...v[n], j) % fà uso di una tabella T(i, j)
1. IF (i==n) {
2.   RETURN j
3.     } ELSE {
4.        IF (T(i, j) non è definito) {
5.           IF (v[i] ≤ j){
6.           T(i, j) = min(Rec_CambioMonete(v[i + 1]...v[n], j), 1 + Rec_CambioMonete(v[i]...v[n], j − v[i]))
              } ELSE {
7.           T(i, j) = Rec_CambioMonete(v[i + 1]...v[n], j)
          }
       }
8. RETURN(T(i, j))
```

Per l'analisi della complessità di tempo di Rec_CambioMonete$(v[1...n], V)$, osserviamo che ciascuna delle nV entrate della matrice $T[\cdot, \cdot]$ viene calcolata una ed una sola volta, ed il tempo necessario per il calcolo di una arbitraria entrata è $O(1)$. Per cui l'algoritmo Rec_CambioMonete$(v[1...n], V)$ ha complessità $O(nV)$.

Sull'esempio con $V = 12$, e monete di valore $v[1] = 10$, $v[2] = 6$, $v[3] = 1$ l'algoritmo costruirebbe la matrice seguente, e produrrebbe in output il valore $C(1, 12) = 2$

								j						
		0	1	2	3	4	5	6	7	8	9	10	11	12
	1	0	1	2	3	4	5	1	2	3	4	1	2	2
i	2	0	1	2	3	4	5	1	2	3	4	5	6	2
	3	0	1	2	3	4	5	6	7	8	9	10	11	12

Il problema del Cambio di Monete è il primo esempio di Problema di Ottmizzazione che abbiamo visto. Informalmente, un Problema di Ottimizzazione è caratterizzato dal fatto che ad ogni possibile istanza di input (ad es., il valore V ed i valori $v[1], \ldots, v[n]$ delle monete nel problema precedente), è possibile associare piú soluzioni (ad es., i diversi modi di esprimere il valore V con le monete di valore $v[1], \ldots, v[n]$). A ciascuna possibile soluzione è associato un costo (ad es., il numero di monete per esprimere V). Ciò che noi cerchiamo è una soluzione di minimo costo (o di massimo costo, se esso rappresenta un "guadagno" per noi). I problemi di ottimizzazione sono quindi caratterizzati dal fatto che ogni istanza di input può avere diverse possibili soluzioni, e noi cerchiamo quella che "ottimizza" il costo (ovvero, lo minimizza o lo massimizza, a seconda dello specifico problema in questione).

◇

Consideriamo ora il seguente esercizio. Dovete affrontare l'esame di Progettazione di Aforismi. L'esame consta di n domande $D_1, \ldots, D_n$ che valgono, rispettivamente, $p[1], \ldots, p[n]$ punti. Ogni domanda D_i richiede $t[i]$ minuti per il suo svolgimento, per ogni $i = 1, \ldots, n$. Il tempo ha disposizione per lo svolgimento dell'esame è T minuti, dove T potrebbe essere inferiore alla somma dei tempi totali $t[1] + \ldots + t[n]$ necessari per rispondere a tutte le domande. Vogliamo progettare un algoritmo efficiente che, dati i vettori $p[1..n]$, $t[1..n]$ ed il valore T, restituisce il punteggio massimo che si può ottenere rispondendo correttamente ad un opportuno sottoinsieme delle n domande entro il tempo massimo di T minuti.

Sia $P[i,j]$ il punteggio massimo che è possibile ottenere rispondendo correttamente ad un opportuno sottoinsieme delle domande $\{D_1, \ldots D_i\}$ avendo a disposizione il tempo massimo di j minuti, per $i = 1, \ldots, n$ e $j = 0, \ldots, T$. La soluzione al problema sarà quindi il valore di $P[n,T]$. É facile vedere che la tabella di programmazione dinamica può essere così definita. Per ogni $j = 0, \ldots T$ si ha

$$P[1,j] = \begin{cases} p[1] & \text{se } j \geq t[1], \\ 0 & \text{altrimenti.} \end{cases}$$

ed, in generale, per $i = 2, \ldots n$, $j = 0, \ldots T$

$$P[i,j] = \begin{cases} \max\{P[i-1,j], P[i-1, j-t[i]] + p[i]\} & \text{se } j \geq t[i], \\ P[i-1,j] & \text{altrimenti.} \end{cases}$$

Abbiamo quindi il seguente algoritmo ricorsivo per il calcolo di `P(n,T)`:

```
Esami(p[1]...p[i], j) % fà uso di una tabella P(i,j)
IF (i==1){
  IF (j>=t[1]) {
   RETURN p[1]
    } ELSE {
     RETURN 0
      } ELSE {
      IF (P(i,j) non è definito) {
        IF (t[i]<= j){
        P(i,j)= max(Esami(p[1]...p[i-1],j-t[i])+p[i], Esami(p[1]...p[i-1],j))
        } ELSE {
          P(i,j)=Esami(p[1]...p[i-1],j)
       }
    }
RETURN (P(i,j))
```

Il tempo richiesto dall'algoritmo è chiaramente pari a $\Theta(nT)$.