PD per il problema dello zaino 0/1 (2/2)

Indice

Un secondo algoritmo di PD
- Numero di passi
- Un'implementazione con liste
- Numero di passi con le liste

Errori tipici

In sintesi

Dove eravamo

- Nella prima parte la ricorrenza $v(j, k)$ (gli oggetti $\{1, 2, \ldots, j\}$ e uno zaino di capacità k) ha dato il valore ottimo del KP.
- Bottom-up, la matrice v si riempie in $\Theta(nc)$ passi; un solo vettore basta per il valore ottimo, e la matrice s delle decisioni dà una soluzione ottima.
- Le liste di stati non dominati (w, p) memorizzano solo gli stati raggiungibili.
- Sulla prima istanza ($c = 5$, profitti $6, 7, 8$, pesi $2, 3, 3$) il valore ottimo è 14, con la soluzione ottima $(1, 0, 1)$.
- **Ora**: un secondo algoritmo di PD, che scambia i ruoli di profitti e pesi.

Un secondo algoritmo di PD

Introduzione: un secondo algoritmo di PD

- Scambiamo i ruoli dei profitti e dei pesi: un sottoproblema è identificato da un profitto obiettivo q
- Il suo valore $u(j, q)$ è il minimo peso totale necessario per ottenere esattamente il profitto q
- Seguiamo lo stesso percorso del primo algoritmo di PD: ricorrenza, top-down, bottom-up, un solo vettore, soluzione ottima, numero di passi e liste di stati
- Il numero di passi diventa $\Theta(nP)$, dove P è il profitto totale degli oggetti: meglio quando P è piccolo

Passo 1: calcolo del valore ottimo

Scambiare profitti e pesi (1/2)

- Nell'algoritmo di PD della sezione precedente, un sottoproblema è identificato da un valore di capacità, e il suo valore ottimo è il massimo profitto totale ottenibile senza superare quella capacità.
- Nell'algoritmo di PD di questa sezione i ruoli dei profitti e dei pesi sono scambiati: un sottoproblema è identificato da un valore di profitto, e il suo valore ottimo è il minimo peso totale necessario per ottenere esattamente quel profitto.
- Poiché ogni profitto p_j è un intero positivo e ogni oggetto si può selezionare al più una volta, il profitto totale di una soluzione ammissibile è un intero compreso fra 0 e il profitto totale di tutti gli oggetti

$$P = \sum_{j=1}^{n} p_j.$$

Scambiare profitti e pesi (2/2)

- Pertanto basta considerare i valori di profitto $q \in \{0, 1, \ldots, P\}$.
- Il numero di sottoproblemi di questo algoritmo di PD cresce con il profitto totale P.
- Per tenere piccolo il numero di sottoproblemi, illustriamo l'algoritmo di PD sulla seguente istanza del KP, che ha gli stessi pesi degli oggetti e la stessa capacità dello zaino della prima istanza, e profitti più piccoli.

Esempio: istanza del KP

Consideriamo uno zaino di capacità $c = 5$ e un insieme $\{1, 2, 3\}$ di 3 oggetti ($n = 3$), con i seguenti profitti e pesi:

	$j = 1$	$j = 2$	$j = 3$
p_j	3	3	4
w_j	2	3	3

Chiamiamo questa istanza del KP la seconda istanza del capitolo.

I sottoproblemi $u(j, q)$

Per ogni $j \in \{0, 1, \ldots, n\}$ e $q \in \{0, 1, \ldots, P\}$, sia $u(j, q)$ il valore ottimo della sottoistanza di peso minimo data da:

1. un profitto obiettivo q,
2. un insieme $\{1, 2, \ldots, j\}$ di oggetti disponibili,
3. per ogni oggetto $h \in \{1, 2, \ldots, j\}$, un profitto p_h e un peso w_h,

che chiede di selezionare un sottoinsieme degli oggetti disponibili il cui profitto totale sia uguale a q, in modo da minimizzare il peso totale degli oggetti selezionati.

La ricorrenza di $u(j, q)$

La formula ricorsiva per calcolare $u(j, q)$ è la seguente:

$$u(j,q) = \begin{cases} 0 & \text{se } q = 0, \\ +\infty & \text{se } j = 0 \\ & \text{e } 1 \leq q \leq P, \\ u(j-1,q) & \text{se } 1 \leq j \leq n, \\ & 1 \leq q \leq P \text{ e } p_j > q, \\ \min \left\{ u(j-1,q), \right. & \text{se } 1 \leq j \leq n, \\ \left. \quad w_j + u(j-1,q-p_j) \right\} & 1 \leq q \leq P,\ p_j \leq q. \end{cases} \tag{1}$$

Lettura della ricorrenza di $u(j, q)$ (1/2)

- Il processo di decomposizione ricorsiva della formula (1) genera al più 2 **sottoproblemi indipendenti**, corrispondenti alla scelta di selezionare o no l'oggetto j per raggiungere il profitto obiettivo q.
- Se $p_j > q$, selezionare l'oggetto j supererebbe il profitto obiettivo e quindi l'unica opzione ammissibile è non selezionarlo.
- Qui $u(j, 0) = 0$ perché il sottoinsieme vuoto ha profitto totale 0 e peso totale 0.
- Inoltre, $u(0, q) = +\infty$ per $q \geq 1$: in questo caso nessun oggetto è disponibile, quindi nessun sottoinsieme ha profitto totale uguale a q. Il valore $+\infty$ serve a escludere questo caso inammissibile.

Lettura della ricorrenza di $u(j, q)$ (2/2)

- Se $p_j \leq q$, il valore ottimo $u(j, q)$ si ottiene provando le due opzioni possibili, selezionare o non selezionare l'oggetto j, e scegliendo quella che minimizza il peso totale.
- I valori corrispondenti sono, rispettivamente, $w_j + u(j-1, q-p_j)$ e $u(j-1, q)$. Se $p_j > q$, l'oggetto j non può essere selezionato e quindi $u(j, q) = u(j-1, q)$.
- Quindi un sottoinsieme di peso minimo con profitto totale q e oggetti disponibili $\{1, 2, \ldots, j\}$ si costruisce combinando la decisione scelta sull'oggetto j con un sottoinsieme di peso minimo del corrispondente sottoproblema residuo.

$\rightsquigarrow$ **Sottostruttura ottima verificata!**

Sottoproblemi sovrapposti

- Sequenze diverse di decisioni sugli oggetti, prese eventualmente per valori diversi di j e q, possono portare allo stesso sottoproblema.
- Quindi lo stesso valore $u(j, q)$ può dover essere calcolato più volte durante la valutazione ricorsiva della formula (1).

$\rightsquigarrow$ **Sottoproblemi sovrapposti verificati!**

Dai valori $u(n, q)$ al valore ottimo

- I valori $u(n, q)$ non danno direttamente il valore ottimo dell'istanza del KP: danno, per ogni profitto obiettivo q, il minimo peso totale necessario per ottenerlo.
- Un sottoinsieme con profitto totale q entra nello zaino se e solo se il suo minimo peso totale non supera la capacità, cioè se e solo se $u(n, q) \leq c$.

Il valore ottimo dell'istanza del KP è

$$\max \left\{ q \in \{0, 1, \ldots, P\} \ : \ u(n, q) \leq c \right\}. \tag{2}$$

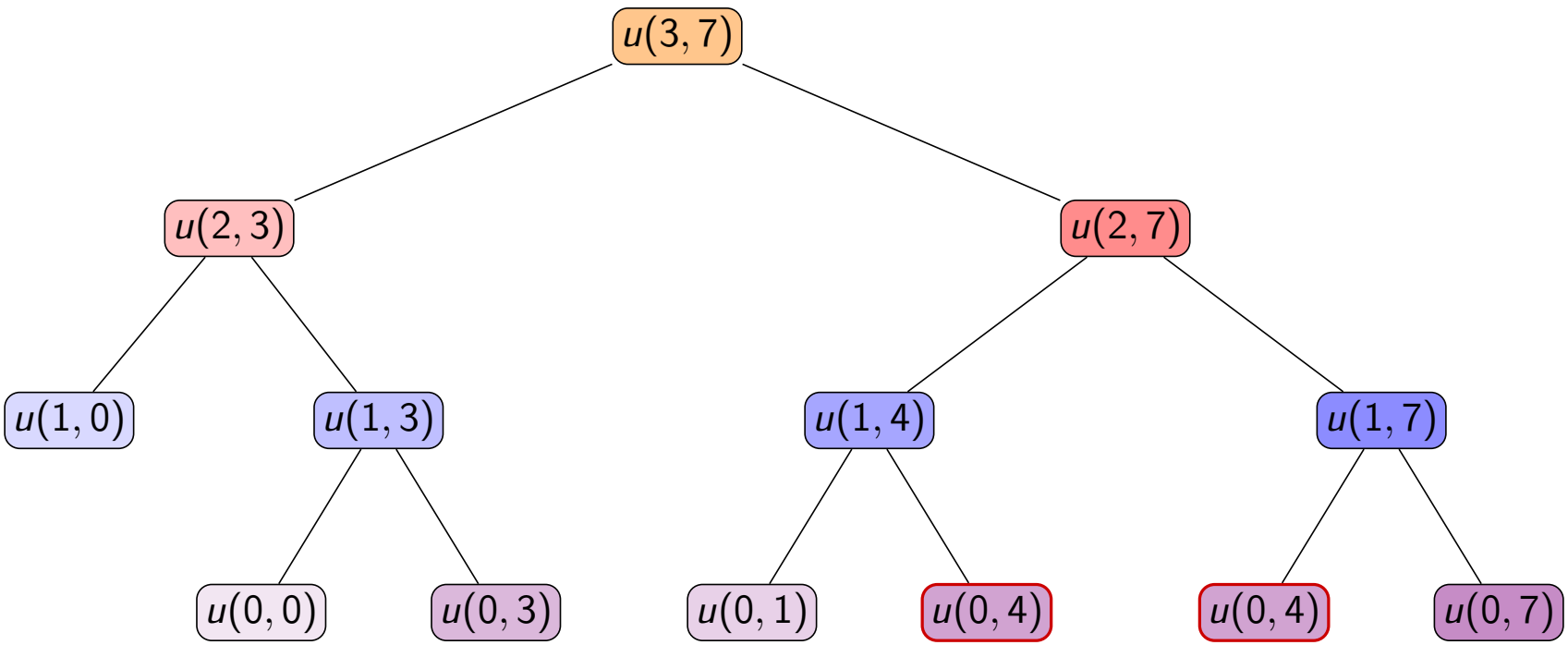
- L'insieme della formula (2) non è mai vuoto, poiché $u(n, 0) = 0 \leq c$, e quindi il massimo è sempre ben definito.

L'approccio top-down

- Un **primo metodo** per calcolare il valore ottimo dell'istanza del KP è l'**approccio top-down**.
- Il metodo valuta ricorsivamente la formula (1), partendo dalla chiamata con $j = n$, per ogni profitto obiettivo $q \in \{0, 1, \ldots, P\}$: ogni chiamata considera l'oggetto j e non lo seleziona oppure, se il suo profitto non supera il profitto obiettivo, considera anche di selezionarlo.
- La ricorsione si ferma quando non restano oggetti o quando il profitto obiettivo è uguale a 0. Infine, si applica la formula (2) ai valori calcolati.
- L'**albero di ricorsione** della funzione u ha un nodo per ogni chiamata ricorsiva, etichettato con la chiamata che rappresenta. La radice rappresenta la chiamata iniziale. I figli di un nodo rappresentano le chiamate ricorsive fatte da quella chiamata. Le foglie rappresentano i casi base.

Esempio: albero di ricorsione (1/2)

Consideriamo la seconda istanza. Questo è l'albero di ricorsione dell'approccio top-down per calcolare $u(3, 7)$:



Esempio: albero di ricorsione (2/2)

- La funzione viene chiamata 13 volte e lo stesso sottoproblema viene risolto più di una volta: $u(0, 4)$ viene chiamata 2 volte (con il bordo rosso).
- Inoltre, l'approccio top-down va applicato a ogni profitto obiettivo $q \in \{0, 1, \ldots, 10\}$, e lo stesso sottoproblema viene generato da chiamate iniziali diverse.
- Per esempio, la chiamata iniziale $u(3, 3)$ genera le chiamate $u(2, 3)$, $u(1, 3)$, $u(1, 0)$, $u(0, 3)$ e $u(0, 0)$, già generate dalla chiamata iniziale $u(3, 7)$ della slide precedente.

Ordine di calcolo

- Il valore $u(j, q)$ si può calcolare quando sono noti i valori richiesti dal membro destro della formula (1).
- In particolare, $u(j-1, q)$ serve quando l'oggetto j non è selezionato e, se $p_j \leq q$, $u(j-1, q-p_j)$ serve quando l'oggetto j è selezionato.
- Pertanto, per un valore fissato di j, tutti i valori $u(j, q)$ si possono calcolare considerando i valori di profitto q in ordine crescente.

Esempio: calcolo dei valori della funzione u (1/3)

Consideriamo la seconda istanza, con uno zaino di capacità $c = 5$ e i seguenti profitti e pesi:

	$j = 1$	$j = 2$	$j = 3$
p_j	3	3	4
w_j	2	3	3

Il profitto totale di tutti gli oggetti è $P = 3 + 3 + 4 = 10$. I valori della funzione u si calcolano come segue:

- I casi base danno $u(0, 0) = 0$ e $u(0, q) = +\infty$ per ogni $q \geq 1$.
- Il primo caso della formula (1) ($q = 0$) dà il caso base:

$$u(1, 0) = 0$$

Esempio: calcolo dei valori della funzione u (2/3)

- Calcolati i casi base, possiamo calcolare $u(1,3)$:

$$u(1,3) = \min\{u(0,3),\ w_1 + u(0,0)\} = \min\{+\infty,\ 2+0\} = 2$$

- Possiamo calcolare anche $u(1,4)$ e $u(1,7)$:

$$u(1,4) = \min\{u(0,4),\ w_1 + u(0,1)\} = \min\{+\infty,\ +\infty\} = +\infty$$

$$u(1,7) = \min\{u(0,7),\ w_1 + u(0,4)\} = \min\{+\infty,\ +\infty\} = +\infty$$

- Calcolati $u(1,3)$ e $u(1,0)$, possiamo calcolare $u(2,3)$:

$$u(2,3) = \min\{u(1,3),\ w_2 + u(1,0)\} = \min\{2,\ 3+0\} = 2$$

Esempio: calcolo dei valori della funzione u (3/3)

- Calcolati $u(1,7)$ e $u(1,4)$, possiamo calcolare $u(2,7)$:

  $$u(2,7) = \min\{u(1,7),\ w_2 + u(1,4)\} = \min\{+\infty,\ +\infty\} = +\infty$$

- Calcolati $u(2,7)$ e $u(2,3)$, possiamo calcolare $u(3,7)$:

  $$u(3,7) = \min\{u(2,7),\ w_3 + u(2,3)\} = \min\{+\infty,\ 3+2\} = 5$$

- Pertanto un profitto totale di 7 si può ottenere con un peso totale di 5, che non supera la capacità $c = 5$ dello zaino.

Lookup table e approccio bottom-up (1/2)

- Possiamo evitare di risolvere ripetutamente gli stessi sottoproblemi memorizzando il valore di ogni sottoproblema appena calcolato. I valori sono memorizzati in una **lookup table**.
- Poiché la soluzione di ogni sottoproblema dipende solo da sottoproblemi con un valore di j più piccolo, possiamo risolvere i sottoproblemi in ordine crescente di j e, per ogni valore fissato di j, in ordine crescente di profitto obiettivo.
- Partiamo dai casi base $u(0,0) = 0$, $u(0,q) = +\infty$ per ogni $q \in \{1,2,\ldots,P\}$ e $u(j,0) = 0$, fino ad arrivare ai valori $u(n,q)$ per ogni $q \in \{0,1,\ldots,P\}$.
- Questa osservazione porta in modo naturale a un **metodo molto più efficiente** per calcolare il valore ottimo dell'istanza del KP.

Lookup table e approccio bottom-up (2/2)

- Un **secondo metodo** per calcolare il valore ottimo dell'istanza del KP è l'**approccio bottom-up**.
- Il metodo parte dai casi base e valuta la formula (1) per valori crescenti di $j \in \{1, 2, \ldots, n\}$ e $q \in \{0, 1, \ldots, P\}$.
- Ogni valore $u(j, q)$ si calcola con valori già calcolati, fino a ottenere tutti i valori $u(n, q)$.
- Il valore ottimo dell'istanza del KP si ottiene poi applicando la formula (2).

Questo è lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**:

Algorithm: DP2

Input : Un'istanza del KP data dal numero n di oggetti, dai vettori $p[1..n]$ e $w[1..n]$ dei profitti e dei pesi degli oggetti e dalla capacità c dello zaino

Output : Il valore ottimo dell'istanza del KP

```
1 function DP2(n, p, w, c)
2     P := 0
3     for j := 1 to n do
4         P ← P + p[j]
5     end
6     u := new array[0..n][0..P]
7     u[0][0] ← 0
```

Algorithm: DP2 (continua)

```
8   for q := 1 to P do
9       u[0][q] ← +∞
10  end
11  for j := 1 to n do
12      for q := 0 to P do
13          if q < p[j] then
14              u[j][q] ← u[j − 1][q]
15          else
16              if w[j] + u[j − 1][q − p[j]] < u[j − 1][q] then
17                  u[j][q] ← w[j] + u[j − 1][q − p[j]]
18              else
19                  u[j][q] ← u[j − 1][q]
20              end
21          end
22      end
23  end
```

Algorithm: DP2 (continua)

```
24      z := 0
25      for q := 0 to P do
26          if u[n][q] ≤ c then
27              z ← q
28          end
29      end
30      return z
31  end
```

Come funziona DP2 (1/2)

- Il primo ciclo calcola il profitto totale P di tutti gli oggetti, che è il più grande profitto obiettivo da considerare.
- La matrice $u[0..n][0..P]$ è la **lookup table**: memorizza i valori ottimi di tutti i sottoproblemi.
- Il secondo ciclo inizializza i valori $u[0][0] = 0$ e $u[0][q] = +\infty$ per tutti i profitti obiettivo $q \in \{1, 2, \ldots, P\}$, corrispondenti al caso in cui nessun oggetto è disponibile.
- Il ciclo esterno, sugli oggetti $j \in \{1, 2, \ldots, n\}$, calcola le righe della matrice in ordine crescente di j.
- Il ciclo interno, su $q \in \{0, 1, \ldots, P\}$, calcola il valore ottimo $u[j][q]$ per ogni profitto obiettivo q.

Come funziona DP2 (2/2)

- Se il profitto dell'oggetto j supera il profitto obiettivo, allora $u[j][q] \leftarrow u[j-1][q]$.
- Altrimenti, l'algoritmo di PD confronta il valore ottenuto non selezionando l'oggetto j, cioè $u[j-1][q]$, con il valore ottenuto selezionando l'oggetto j, cioè $w[j] + u[j-1][q-p[j]]$.
- La variabile ausiliaria z memorizza il più grande profitto obiettivo il cui minimo peso totale non supera la capacità dello zaino. Si inizializza a 0, che è sempre un profitto obiettivo ammissibile poiché $u[n][0] = 0 \leq c$.
- L'ultimo ciclo scorre i profitti obiettivo in ordine crescente e aggiorna z ogni volta che $u[n][q] \leq c$, in modo che alla fine z sia il massimo della formula (2).
- Infine, l'algoritmo restituisce il valore ottimo dell'istanza del KP, z.

Esempio: esecuzione dell'algoritmo di PD (1/7)

Consideriamo la seconda istanza, con uno zaino di capacità $c = 5$ e i seguenti profitti e pesi:

	$j = 1$	$j = 2$	$j = 3$
p_j	3	3	4
w_j	2	3	3

- Il profitto totale di tutti gli oggetti è $P = 3 + 3 + 4 = 10$.
- Per ogni passo dell'algoritmo di PD, le tabelle delle slide successive riportano i valori di j e q, la cella $u[j][q]$ e l'operazione eseguita.

Esempio: esecuzione dell'algoritmo di PD (2/7)

j	q	$u[j][q]$	Azione
0	0	0	$u[0][0] \leftarrow 0$
0	1	$+\infty$	$u[0][1] \leftarrow +\infty$
0	2	$+\infty$	$u[0][2] \leftarrow +\infty$
0	3	$+\infty$	$u[0][3] \leftarrow +\infty$
0	4	$+\infty$	$u[0][4] \leftarrow +\infty$
0	5	$+\infty$	$u[0][5] \leftarrow +\infty$
0	6	$+\infty$	$u[0][6] \leftarrow +\infty$
0	7	$+\infty$	$u[0][7] \leftarrow +\infty$
0	8	$+\infty$	$u[0][8] \leftarrow +\infty$
0	9	$+\infty$	$u[0][9] \leftarrow +\infty$
0	10	$+\infty$	$u[0][10] \leftarrow +\infty$

Esempio: esecuzione dell'algoritmo di PD (3/7)

j	q	$u[j][q]$	Azione
1	0	0	$u[1][0] \leftarrow u[0][0] = 0$
1	1	$+\infty$	$u[1][1] \leftarrow u[0][1] = +\infty$
1	2	$+\infty$	$u[1][2] \leftarrow u[0][2] = +\infty$
1	3	2	$\min\{u[0][3], w[1] + u[0][0]\} = \min\{+\infty, 2 + 0\} = 2$
1	4	$+\infty$	$\min\{u[0][4], w[1] + u[0][1]\} = \min\{+\infty, +\infty\} = +\infty$
1	5	$+\infty$	$\min\{u[0][5], w[1] + u[0][2]\} = \min\{+\infty, +\infty\} = +\infty$
1	6	$+\infty$	$\min\{u[0][6], w[1] + u[0][3]\} = \min\{+\infty, +\infty\} = +\infty$
1	7	$+\infty$	$\min\{u[0][7], w[1] + u[0][4]\} = \min\{+\infty, +\infty\} = +\infty$
1	8	$+\infty$	$\min\{u[0][8], w[1] + u[0][5]\} = \min\{+\infty, +\infty\} = +\infty$
1	9	$+\infty$	$\min\{u[0][9], w[1] + u[0][6]\} = \min\{+\infty, +\infty\} = +\infty$
1	10	$+\infty$	$\min\{u[0][10], w[1] + u[0][7]\} = \min\{+\infty, +\infty\} = +\infty$

Esempio: esecuzione dell'algoritmo di PD (4/7)

j	q	$u[j][q]$	Azione
2	0	0	$u[2][0] \leftarrow u[1][0] = 0$
2	1	$+\infty$	$u[2][1] \leftarrow u[1][1] = +\infty$
2	2	$+\infty$	$u[2][2] \leftarrow u[1][2] = +\infty$
2	3	2	$\min\{u[1][3], w[2] + u[1][0]\} = \min\{2, 3 + 0\} = 2$
2	4	$+\infty$	$\min\{u[1][4], w[2] + u[1][1]\} = \min\{+\infty, +\infty\} = +\infty$
2	5	$+\infty$	$\min\{u[1][5], w[2] + u[1][2]\} = \min\{+\infty, +\infty\} = +\infty$
2	6	5	$\min\{u[1][6], w[2] + u[1][3]\} = \min\{+\infty, 3 + 2\} = 5$
2	7	$+\infty$	$\min\{u[1][7], w[2] + u[1][4]\} = \min\{+\infty, +\infty\} = +\infty$
2	8	$+\infty$	$\min\{u[1][8], w[2] + u[1][5]\} = \min\{+\infty, +\infty\} = +\infty$
2	9	$+\infty$	$\min\{u[1][9], w[2] + u[1][6]\} = \min\{+\infty, +\infty\} = +\infty$
2	10	$+\infty$	$\min\{u[1][10], w[2] + u[1][7]\} = \min\{+\infty, +\infty\} = +\infty$

Esempio: esecuzione dell'algoritmo di PD (5/7)

j	q	$u[j][q]$	Azione
3	0	0	$u[3][0] \leftarrow u[2][0] = 0$
3	1	$+\infty$	$u[3][1] \leftarrow u[2][1] = +\infty$
3	2	$+\infty$	$u[3][2] \leftarrow u[2][2] = +\infty$
3	3	2	$u[3][3] \leftarrow u[2][3] = 2$
3	4	3	$\min\{u[2][4], w[3] + u[2][0]\} = \min\{+\infty, 3 + 0\} = 3$
3	5	$+\infty$	$\min\{u[2][5], w[3] + u[2][1]\} = \min\{+\infty, +\infty\} = +\infty$
3	6	5	$\min\{u[2][6], w[3] + u[2][2]\} = \min\{5, +\infty\} = 5$
3	7	5	$\min\{u[2][7], w[3] + u[2][3]\} = \min\{+\infty, 3 + 2\} = 5$
3	8	$+\infty$	$\min\{u[2][8], w[3] + u[2][4]\} = \min\{+\infty, +\infty\} = +\infty$
3	9	$+\infty$	$\min\{u[2][9], w[3] + u[2][5]\} = \min\{+\infty, +\infty\} = +\infty$
3	10	8	$\min\{u[2][10], w[3] + u[2][6]\} = \min\{+\infty, 3 + 5\} = 8$

Esempio: esecuzione dell'algoritmo di PD (6/7)

Dopo l'esecuzione dei due cicli annidati dell'algoritmo di PD, questa è la matrice u:

$u[j][q]$	0	1	2	3	4	5	6	7	8	9	10
$j = 0$	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$j = 1$	0	$+\infty$	$+\infty$	2	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$j = 2$	0	$+\infty$	$+\infty$	2	$+\infty$	$+\infty$	5	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$j = 3$	0	$+\infty$	$+\infty$	2	3	$+\infty$	5	5	$+\infty$	$+\infty$	8

Esempio: esecuzione dell'algoritmo di PD (7/7)

L'ultimo ciclo dell'algoritmo di PD scorre l'ultima riga della matrice u e tiene il più grande profitto obiettivo il cui minimo peso totale non supera la capacità $c = 5$ dello zaino:

q	$u[3][q]$	Azione
0	0	$u[3][0] = 0 \leq 5$: $z \leftarrow 0$
1	$+\infty$	$u[3][1] = +\infty > 5$: nessun aggiornamento
2	$+\infty$	$u[3][2] = +\infty > 5$: nessun aggiornamento
3	2	$u[3][3] = 2 \leq 5$: $z \leftarrow 3$
4	3	$u[3][4] = 3 \leq 5$: $z \leftarrow 4$
5	$+\infty$	$u[3][5] = +\infty > 5$: nessun aggiornamento
6	5	$u[3][6] = 5 \leq 5$: $z \leftarrow 6$
7	5	$u[3][7] = 5 \leq 5$: $z \leftarrow 7$
8	$+\infty$	$u[3][8] = +\infty > 5$: nessun aggiornamento
9	$+\infty$	$u[3][9] = +\infty > 5$: nessun aggiornamento
10	8	$u[3][10] = 8 > 5$: nessun aggiornamento

Quindi il valore ottimo dell'istanza del KP è $z = 7$.

Dalla matrice a un solo vettore

- Dopo l'esecuzione dell'algoritmo di PD, la matrice u contiene il valore ottimo $u[j][q] = u(j, q)$ per ogni oggetto $j \in \{0, 1, \ldots, n\}$ e ogni profitto obiettivo $q \in \{0, 1, \ldots, P\}$.
- L'algoritmo di PD calcola la riga j della matrice u usando solo la riga $j - 1$.
- Pertanto, quando serve solo il valore ottimo dell'istanza del KP, non è necessario memorizzare tutta la matrice: basta un solo vettore $u[0..P]$, che usa $P + 1$ celle invece delle $(n + 1)(P + 1)$ celle della matrice.

Un solo vettore (1/2)

- Il vettore si inizializza con i casi base $u[0] = 0$ e $u[q] = +\infty$ per ogni $q \in \{1, 2, \ldots, P\}$, che corrispondono alla riga $j = 0$ della matrice.
- Poi, per ogni oggetto $j \in \{1, 2, \ldots, n\}$, il vettore viene aggiornato in modo da contenere la riga j della matrice.
- L'ordine di aggiornamento delle celle è essenziale. Quando la cella $u[q]$ viene aggiornata per l'oggetto j, il valore $u[q - p_j]$ deve essere ancora quello della riga $j - 1$, cioè $u(j - 1, q - p_j)$.

Un solo vettore (2/2)

- Poiché $q - p_j < q$, questo è garantito se i profitti obiettivo sono considerati in ordine **decrescente** $q = P, P-1, \ldots, p_j$: quando la cella q viene aggiornata, la cella $q - p_j$ non è ancora stata aggiornata nella passata corrente.
- Se i profitti obiettivo fossero considerati in ordine crescente, il valore $u[q - p_j]$ sarebbe già quello della riga j, cioè $u(j, q - p_j)$, e l'oggetto j potrebbe essere selezionato più di una volta, cosa non consentita nel KP.
- Per ogni profitto obiettivo $q < p_j$, il profitto dell'oggetto j supera il profitto obiettivo e quindi $u[q]$ deve mantenere il valore $u(j-1, q)$: non si esegue alcun aggiornamento e il ciclo interno può fermarsi a $q = p_j$.

Questo è lo pseudocodice dell'**algoritmo di PD con un solo vettore**:

Algorithm: DP2_Vector

Input : Un'istanza del KP data dal numero n di oggetti, dai vettori $p[1..n]$ e $w[1..n]$ dei profitti e dei pesi degli oggetti e dalla capacità c dello zaino

Output : Il valore ottimo dell'istanza del KP

```
1 function DP2_Vector(n, p, w, c)
2     P := 0
3     for j := 1 to n do
4         P ← P + p[j]
5     end
6     u := new array[0..P]
7     u[0] ← 0
```

Algorithm: DP2_Vector (continua)

```
8   |  for q := 1 to P do
9   |  |  u[q] ← +∞
10  |  end
11  |  for j := 1 to n do
12  |  |  for q := P downto p[j] do
13  |  |  |  if w[j] + u[q − p[j]] < u[q] then
14  |  |  |  |  u[q] ← w[j] + u[q − p[j]]
15  |  |  |  end
16  |  |  end
17  |  end
```

Algorithm: DP2_Vector (continua)

```
18      z := 0
19      for q := 0 to P do
20          if u[q] ≤ c then
21              z ← q
22          end
23      end
24      return z
25  end
```

Come funziona DP2_Vector (1/2)

- Il vettore $u[0..P]$ è la **lookup table**: al termine della passata per l'oggetto j, memorizza la riga j della matrice u dell'algoritmo di PD.
- Il secondo ciclo inizializza i valori $u[0] = 0$ e $u[q] = +\infty$ per tutti i profitti obiettivo $q \in \{1, 2, \ldots, P\}$, corrispondenti al caso in cui nessun oggetto è disponibile.
- Il ciclo esterno, sugli oggetti $j \in \{1, 2, \ldots, n\}$, esegue una passata del vettore per ogni oggetto.

Come funziona DP2_Vector (2/2)

- Il ciclo interno, su $q \in \{p[j], p[j]+1, \ldots, P\}$, considera i profitti obiettivo in ordine decrescente, in modo che le celle che servono ancora con il valore precedente non siano ancora state modificate.
- La condizione confronta il valore ottenuto non selezionando l'oggetto j, cioè il valore corrente $u[q]$, con il valore ottenuto selezionando l'oggetto j, cioè $w[j] + u[q - p[j]]$. La cella viene aggiornata solo quando selezionare l'oggetto j è strettamente migliore.
- L'ultimo ciclo scorre il vettore e restituisce il più grande profitto obiettivo il cui minimo peso totale non supera la capacità dello zaino, come nella formula (2).

Esempio: esecuzione di DP2 con un solo vettore (1/5)

Consideriamo la seconda istanza, con uno zaino di capacità $c = 5$ e i seguenti profitti e pesi:

	$j = 1$	$j = 2$	$j = 3$
p_j	3	3	4
w_j	2	3	3

- Il profitto totale di tutti gli oggetti è $P = 3 + 3 + 4 = 10$ e il secondo ciclo inizializza $u[0] = 0$ e $u[q] = +\infty$ per ogni $q \in \{1, 2, \ldots, 10\}$.
- Per ogni passo dei due cicli annidati, le tabelle delle slide successive riportano i valori di j e q, la cella $u[q]$ dopo il passo e l'operazione eseguita.

Esempio: esecuzione di DP2 con un solo vettore (2/5)

j	q	$u[q]$	Azione
1	10	$+\infty$	$w[1] + u[7] = +\infty \geq u[10] = +\infty$: nessun aggiornamento
1	9	$+\infty$	$w[1] + u[6] = +\infty \geq u[9] = +\infty$: nessun aggiornamento
1	8	$+\infty$	$w[1] + u[5] = +\infty \geq u[8] = +\infty$: nessun aggiornamento
1	7	$+\infty$	$w[1] + u[4] = +\infty \geq u[7] = +\infty$: nessun aggiornamento
1	6	$+\infty$	$w[1] + u[3] = +\infty \geq u[6] = +\infty$: nessun aggiornamento
1	5	$+\infty$	$w[1] + u[2] = +\infty \geq u[5] = +\infty$: nessun aggiornamento
1	4	$+\infty$	$w[1] + u[1] = +\infty \geq u[4] = +\infty$: nessun aggiornamento
1	3	2	$w[1] + u[0] = 2 < u[3] = +\infty$: $u[3] \leftarrow 2$

Esempio: esecuzione di DP2 con un solo vettore (3/5)

j	q	$u[q]$	Azione
2	10	$+\infty$	$w[2]+u[7]=+\infty \geq u[10]=+\infty$: nessun aggiornamento
2	9	$+\infty$	$w[2]+u[6]=+\infty \geq u[9]=+\infty$: nessun aggiornamento
2	8	$+\infty$	$w[2]+u[5]=+\infty \geq u[8]=+\infty$: nessun aggiornamento
2	7	$+\infty$	$w[2]+u[4]=+\infty \geq u[7]=+\infty$: nessun aggiornamento
2	6	5	$w[2]+u[3]=5 < u[6]=+\infty$: $u[6] \leftarrow 5$
2	5	$+\infty$	$w[2]+u[2]=+\infty \geq u[5]=+\infty$: nessun aggiornamento
2	4	$+\infty$	$w[2]+u[1]=+\infty \geq u[4]=+\infty$: nessun aggiornamento
2	3	2	$w[2]+u[0]=3 \geq u[3]=2$: nessun aggiornamento

Esempio: esecuzione di DP2 con un solo vettore (4/5)

j	q	$u[q]$	Azione
3	10	8	$w[3]+u[6]=8<u[10]=+\infty$: $u[10]\leftarrow 8$
3	9	$+\infty$	$w[3]+u[5]=+\infty\geq u[9]=+\infty$: nessun aggiornamento
3	8	$+\infty$	$w[3]+u[4]=+\infty\geq u[8]=+\infty$: nessun aggiornamento
3	7	5	$w[3]+u[3]=5<u[7]=+\infty$: $u[7]\leftarrow 5$
3	6	5	$w[3]+u[2]=+\infty\geq u[6]=5$: nessun aggiornamento
3	5	$+\infty$	$w[3]+u[1]=+\infty\geq u[5]=+\infty$: nessun aggiornamento
3	4	3	$w[3]+u[0]=3<u[4]=+\infty$: $u[4]\leftarrow 3$

Esempio: esecuzione di DP2 con un solo vettore (5/5)

Questo è il contenuto del vettore u dopo l'inizializzazione e dopo ogni passata del ciclo esterno:

$u[q]$	0	1	2	3	4	5	6	7	8	9	10
inizializzazione	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
passata $j = 1$	0	$+\infty$	$+\infty$	2	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
passata $j = 2$	0	$+\infty$	$+\infty$	2	$+\infty$	$+\infty$	5	$+\infty$	$+\infty$	$+\infty$	$+\infty$
passata $j = 3$	0	$+\infty$	$+\infty$	2	3	$+\infty$	5	5	$+\infty$	$+\infty$	8

Ogni passata trasforma il vettore u dalla riga $j - 1$ nella riga j della matrice u calcolata dall'algoritmo di PD sulla stessa istanza.
L'ultimo ciclo scorre poi il vettore e dà il valore ottimo $z = 7$ dell'istanza del KP, come con la matrice.

Il vettore calcola solo il valore ottimo

- La versione con un vettore calcola solo il valore ottimo dell'istanza del KP.
- L'informazione necessaria per costruire una soluzione ottima, descritta nel Passo 2, registra la decisione presa per ogni oggetto j e ogni profitto obiettivo q, e quindi non si può memorizzare in un solo vettore.

Passo 2: calcolo di una soluzione ottima

La matrice s delle decisioni

- Per calcolare una soluzione ottima, cioè il vettore binario $(x_1, x_2, \ldots, x_n)$, è necessario memorizzare alcune informazioni aggiuntive.
- Per ogni oggetto j e profitto obiettivo q, oltre al valore ottimo $u[j][q]$, memorizziamo $s[j][q]$, che indica se l'oggetto j è selezionato in un sottoinsieme di peso minimo degli oggetti $\{1, 2, \ldots, j\}$ con profitto totale uguale a q.
- Più precisamente, $s[j][q]$ è uguale a 1 se l'oggetto j è selezionato nel calcolo del valore $u[j][q]$, ed è uguale a 0 altrimenti.

Questo è lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**, con la memorizzazione delle informazioni necessarie per costruire una soluzione ottima:

Algorithm: DP2_InfoSol

Input : Un'istanza del KP data dal numero n di oggetti, dai vettori $p[1..n]$ e $w[1..n]$ dei profitti e dei pesi degli oggetti e dalla capacità c dello zaino

Output : Il valore ottimo dell'istanza del KP e la matrice delle informazioni binarie $s[0..n][0..P]$

```
1 function DP2_InfoSol(n, p, w, c)
2 |   P := 0
3 |   for j := 1 to n do
4 |   |   P ← P + p[j]
5 |   end
```

Algorithm: DP2_InfoSol (continua)

```
6   u := new array[0..n][0..P]
7   s := new array[0..n][0..P]
8   u[0][0] ← 0
9   s[0][0] ← 0
10  for q := 1 to P do
11      u[0][q] ← +∞
12      s[0][q] ← 0
13  end
```

Algorithm: DP2_InfoSol (continua)

```
14  for j := 1 to n do
15      for q := 0 to P do
16          if q < p[j] then
17              u[j][q] ← u[j − 1][q]
18              s[j][q] ← 0
19          else
20              if w[j] + u[j − 1][q − p[j]] < u[j − 1][q] then
21                  u[j][q] ← w[j] + u[j − 1][q − p[j]]
22                  s[j][q] ← 1
23              else
24                  u[j][q] ← u[j − 1][q]
25                  s[j][q] ← 0
26              end
27          end
28      end
29  end
```

Algorithm: DP2_InfoSol (continua)

```
30      z := 0
31      for q := 0 to P do
32          if u[n][q] ≤ c then
33              z ← q
34          end
35      end
36      return z, s
37  end
```

- L'algoritmo di PD usa la matrice aggiuntiva s, che registra la decisione presa quando si calcola $u[j][q]$. Se l'oggetto j è selezionato, $s[j][q] \leftarrow 1$ memorizza questa decisione; altrimenti, $s[j][q] \leftarrow 0$ memorizza che l'oggetto j non è selezionato.

Esempio: calcolo delle informazioni aggiuntive (1/6)

Consideriamo la seconda istanza, con uno zaino di capacità $c = 5$ e i seguenti profitti e pesi:

	$j = 1$	$j = 2$	$j = 3$
p_j	3	3	4
w_j	2	3	3

Per ogni passo dell'algoritmo di PD, le tabelle delle slide successive riportano i valori di j e q, la cella $s[j][q]$ e l'operazione eseguita.

Esempio: calcolo delle informazioni aggiuntive (2/6)

j	q	$s[j][q]$	Azione
0	0	0	$s[0][0] \leftarrow 0$
0	1	0	$s[0][1] \leftarrow 0$
0	2	0	$s[0][2] \leftarrow 0$
0	3	0	$s[0][3] \leftarrow 0$
0	4	0	$s[0][4] \leftarrow 0$
0	5	0	$s[0][5] \leftarrow 0$
0	6	0	$s[0][6] \leftarrow 0$
0	7	0	$s[0][7] \leftarrow 0$
0	8	0	$s[0][8] \leftarrow 0$
0	9	0	$s[0][9] \leftarrow 0$
0	10	0	$s[0][10] \leftarrow 0$

Esempio: calcolo delle informazioni aggiuntive (3/6)

j	q	$s[j][q]$	Azione
1	0	0	$0 < 3$: $s[1][0] \leftarrow 0$
1	1	0	$1 < 3$: $s[1][1] \leftarrow 0$
1	2	0	$2 < 3$: $s[1][2] \leftarrow 0$
1	3	1	$w[1] + u[0][0] = 2 + 0 < u[0][3] = +\infty$: $s[1][3] \leftarrow 1$
1	4	0	$w[1] + u[0][1] = +\infty \geq u[0][4] = +\infty$: $s[1][4] \leftarrow 0$
1	5	0	$w[1] + u[0][2] = +\infty \geq u[0][5] = +\infty$: $s[1][5] \leftarrow 0$
1	6	0	$w[1] + u[0][3] = +\infty \geq u[0][6] = +\infty$: $s[1][6] \leftarrow 0$
1	7	0	$w[1] + u[0][4] = +\infty \geq u[0][7] = +\infty$: $s[1][7] \leftarrow 0$
1	8	0	$w[1] + u[0][5] = +\infty \geq u[0][8] = +\infty$: $s[1][8] \leftarrow 0$
1	9	0	$w[1] + u[0][6] = +\infty \geq u[0][9] = +\infty$: $s[1][9] \leftarrow 0$
1	10	0	$w[1] + u[0][7] = +\infty \geq u[0][10] = +\infty$: $s[1][10] \leftarrow 0$

Esempio: calcolo delle informazioni aggiuntive (4/6)

j	q	$s[j][q]$	Azione
2	0	0	$0 < 3$: $s[2][0] \leftarrow 0$
2	1	0	$1 < 3$: $s[2][1] \leftarrow 0$
2	2	0	$2 < 3$: $s[2][2] \leftarrow 0$
2	3	0	$w[2] + u[1][0] = 3 + 0 \geq u[1][3] = 2$: $s[2][3] \leftarrow 0$
2	4	0	$w[2] + u[1][1] = +\infty \geq u[1][4] = +\infty$: $s[2][4] \leftarrow 0$
2	5	0	$w[2] + u[1][2] = +\infty \geq u[1][5] = +\infty$: $s[2][5] \leftarrow 0$
2	6	1	$w[2] + u[1][3] = 3 + 2 < u[1][6] = +\infty$: $s[2][6] \leftarrow 1$
2	7	0	$w[2] + u[1][4] = +\infty \geq u[1][7] = +\infty$: $s[2][7] \leftarrow 0$
2	8	0	$w[2] + u[1][5] = +\infty \geq u[1][8] = +\infty$: $s[2][8] \leftarrow 0$
2	9	0	$w[2] + u[1][6] = +\infty \geq u[1][9] = +\infty$: $s[2][9] \leftarrow 0$
2	10	0	$w[2] + u[1][7] = +\infty \geq u[1][10] = +\infty$: $s[2][10] \leftarrow 0$

Esempio: calcolo delle informazioni aggiuntive (5/6)

j	q	$s[j][q]$	Azione
3	0	0	$0 < 4$: $s[3][0] \leftarrow 0$
3	1	0	$1 < 4$: $s[3][1] \leftarrow 0$
3	2	0	$2 < 4$: $s[3][2] \leftarrow 0$
3	3	0	$3 < 4$: $s[3][3] \leftarrow 0$
3	4	1	$w[3] + u[2][0] = 3 + 0 < u[2][4] = +\infty$: $s[3][4] \leftarrow 1$
3	5	0	$w[3] + u[2][1] = +\infty \geq u[2][5] = +\infty$: $s[3][5] \leftarrow 0$
3	6	0	$w[3] + u[2][2] = +\infty \geq u[2][6] = 5$: $s[3][6] \leftarrow 0$
3	7	1	$w[3] + u[2][3] = 3 + 2 < u[2][7] = +\infty$: $s[3][7] \leftarrow 1$
3	8	0	$w[3] + u[2][4] = +\infty \geq u[2][8] = +\infty$: $s[3][8] \leftarrow 0$
3	9	0	$w[3] + u[2][5] = +\infty \geq u[2][9] = +\infty$: $s[3][9] \leftarrow 0$
3	10	1	$w[3] + u[2][6] = 3 + 5 < u[2][10] = +\infty$: $s[3][10] \leftarrow 1$

Esempio: calcolo delle informazioni aggiuntive (6/6)

Questa è la matrice s dopo l'esecuzione dell'algoritmo di PD:

$s[j][q]$	0	1	2	3	4	5	6	7	8	9	10
$j = 0$	0	0	0	0	0	0	0	0	0	0	0
$j = 1$	0	0	0	1	0	0	0	0	0	0	0
$j = 2$	0	0	0	0	0	0	1	0	0	0	0
$j = 3$	0	0	0	0	1	0	0	1	0	0	1

Ricostruzione di una soluzione ottima

- Calcolata la matrice s, si può ricostruire una soluzione ottima scorrendo gli oggetti da $j = n$ fino a $j = 1$, partendo dal profitto obiettivo $q = z$ restituito dall'algoritmo di PD e da $x_j = 0$ per ogni $j \in \{1, 2, \ldots, n\}$:

$$\begin{aligned} x_j \leftarrow 1, \quad q \leftarrow q - p_j \quad & \text{se } s[j][q] = 1, \\ x_j \leftarrow 0 \quad & \text{se } s[j][q] = 0. \end{aligned}$$

- Si ottiene così il vettore $(x_1, x_2, \ldots, x_n)$, dove x_j è uguale a 1 se l'oggetto j è selezionato in una soluzione ottima, e a 0 altrimenti.

Algorithm: DP2_Reconstruct

Input : Il numero n di oggetti, il valore ottimo z dell'istanza del KP, il vettore $p[1..n]$ dei profitti degli oggetti e la matrice $s[0..n][0..P]$

Output : Una soluzione ottima dell'istanza del KP

```
1 function DP2_Reconstruct(n, z, p, s)
2 |   x := new array[1..n]
3 |   for j := 1 to n do
4 |   |   x[j] ← 0
5 |   end
6 |   q := z
```

Algorithm: DP2_Reconstruct (continua)

```
7     for j := n downto 1 do
8         if s[j][q] = 1 then
9             x[j] ← 1
10            q ← q − p[j]
11        end
12    end
13    return x
14 end
```

Esempio: calcolo di una soluzione ottima (1/2)

Dopo l'esecuzione dell'algoritmo di PD sulla seconda istanza, il valore ottimo è $z = 7$ e abbiamo la seguente matrice s:

$s[j][q]$	0	1	2	3	4	5	6	7	8	9	10
$j = 0$	0	0	0	0	0	0	0	0	0	0	0
$j = 1$	0	0	0	1	0	0	0	0	0	0	0
$j = 2$	0	0	0	0	0	0	1	0	0	0	0
$j = 3$	0	0	0	0	1	0	0	1	0	0	1

Esempio: calcolo di una soluzione ottima (2/2)

Possiamo ricostruire una soluzione ottima partendo da $j = 3$ e $q = z = 7$ come segue:

Iterazione	j	q	$s[j][q]$	$x[1]$	$x[2]$	$x[3]$	Azione
–	–	–	–	0	0	0	$q = 7$
1	3	7	1	0	0	1	$q \leftarrow 7 - 4 = 3$
2	2	3	0	0	0	1	nessun aggiornamento
3	1	3	1	1	0	1	$q \leftarrow 3 - 3 = 0$

Quindi il vettore

$$(1, 0, 1)$$

è una soluzione ottima. Il valore ottimo corrispondente è $3 + 4 = 7$.

Quanti passi esegue il secondo algoritmo di PD?

Contiamo quante volte viene eseguita ogni riga dello pseudocodice di `DP2`.

Numero di passi di DP2 (1/5)

- Indichiamo con $S(n, P)$ il numero di passi eseguiti da `DP2` nel caso peggiore, dove n è il numero di oggetti e $P = \sum_{j=1}^{n} p_j$ è il profitto totale di tutti gli oggetti.
- Calcoliamo $S(n, P)$ contando quante volte vengono eseguite le righe dello pseudocodice di `DP2`.
- La condizione della riga 16 viene verificata solo quando $q \geq p[j]$, cioè per $P + 1 - p_j$ profitti obiettivo per ogni oggetto j, e quindi $n(P + 1) - P$ volte in tutto.

Numero di passi di DP2 (2/5)

Il numero di esecuzioni delle righe rilevanti è:

- La Riga 2 viene eseguita 1 volta.
- La Riga 3 viene eseguita n volte.
- La Riga 4 viene eseguita n volte.
- La Riga 6 viene eseguita 1 volta.
- La Riga 7 viene eseguita 1 volta.
- La Riga 8 viene eseguita P volte.
- La Riga 9 viene eseguita P volte.
- La Riga 11 viene eseguita n volte.
- La Riga 12 viene eseguita $n(P+1)$ volte.
- La Riga 13 viene eseguita $n(P+1)$ volte.

Numero di passi di DP2 (3/5)

- La Riga 16 viene eseguita $n(P+1)-P$ volte.
- una fra la Riga 14, la Riga 17 e la Riga 19 viene eseguita per ogni coppia (j,q), e quindi queste righe di assegnamento vengono eseguite in tutto $n(P+1)$ volte.
- La Riga 24 viene eseguita 1 volta.
- La Riga 25 viene eseguita $P+1$ volte.
- La Riga 26 viene eseguita $P+1$ volte.
- La Riga 27 viene eseguita al più $P+1$ volte.
- La Riga 30 viene eseguita 1 volta.

Numero di passi di DP2 (4/5)

Pertanto, il numero totale di passi nel caso peggiore è

$$\begin{aligned}&1 + n + n + 1 + 1 + P + P + n + n(P+1) + n(P+1)\\&+ \big(n(P+1) - P\big) + n(P+1) + 1\\&+ (P+1) + (P+1) + (P+1) + 1.\end{aligned}$$

Quindi

$$S(n, P) = 4nP + 7n + 4P + 8.$$

Dunque

$$S(n, P) = \Theta(nP).$$

Numero di passi di DP2 (5/5)

Infatti

$$\lim_{n,P\to\infty} \frac{4nP + 7n + 4P + 8}{nP} = \lim_{n,P\to\infty} \left(4 + \frac{7}{P} + \frac{4}{n} + \frac{8}{nP}\right)$$
$$= 4 \in \mathbb{R}_{>0}.$$

Il numero di passi delle tre versioni

L'algoritmo di PD esegue

$$S(n, P) = \Theta(nP)$$

passi nel caso peggiore, dove $P = \sum_{j=1}^{n} p_j$.

- La versione `DP2_InfoSol` memorizza la matrice aggiuntiva s ed esegue solo un numero costante di operazioni in più dentro gli stessi cicli. Pertanto, il suo numero di passi resta $\Theta(nP)$.
- La versione `DP2_Vector` usa un solo vettore invece della matrice ed esegue gli stessi due cicli annidati, il cui ciclo interno esegue al più $P + 1$ iterazioni per ogni oggetto. Pertanto, il suo numero di passi resta $\Theta(nP)$, mentre il numero di celle della lookup table scende da $(n + 1)(P + 1)$ a $P + 1$.

Un'implementazione con liste

Passo 1: calcolo del valore ottimo

Perché le liste?

- L'algoritmo di PD di questa sezione memorizza un valore per ogni coppia formata da un oggetto e da un profitto obiettivo, cioè $(n+1)(P+1)$ valori in tutto.
- Molti di questi valori sono uguali a $+\infty$, perché nessun sottoinsieme degli oggetti disponibili ha quel profitto totale.
- Vediamo ora una **diversa implementazione dello stesso algoritmo di PD**. La decomposizione ricorsiva e la sottostruttura ottima sono quelle della formula (1): cambia la struttura dati.
- Invece di un'intera riga della matrice, memorizziamo solo le coppie ottenute effettivamente da un sottoinsieme di oggetti e, fra queste, solo quelle che possono portare a una soluzione ottima. Queste coppie sono tenute in una lista, una lista per ogni oggetto.

Stati

Uno **stato** è una coppia (w, p), dove w è il peso totale e p è il profitto totale di un sottoinsieme di oggetti.

- Il sottoinsieme vuoto dà lo stato $(0, 0)$. Un sottoinsieme degli oggetti $\{1, 2, \ldots, j\}$ dà lo stato ottenuto sommando i pesi e i profitti dei suoi oggetti.
- Due sottoinsiemi diversi possono dare lo stesso stato, e in questo caso basta tenerne uno.
- Più in generale, uno stato si può scartare quando un altro stato è almeno altrettanto buono in entrambe le componenti.

Stati dominati

Dati due stati (w, p) e (w', p') con $(w, p) \neq (w', p')$, lo stato (w, p) **domina** lo stato (w', p') se

$$w \leq w' \qquad \text{e} \qquad p \geq p'.$$

Uno stato è **non dominato** se nessun altro stato lo domina.

- Uno stato dominato si può scartare. Infatti, supponiamo che (w, p) domini (w', p') e sia S' un sottoinsieme degli oggetti rimanenti che si può aggiungere a (w', p') senza superare la capacità.
- Poiché $w \leq w'$, lo stesso sottoinsieme S' si può aggiungere anche a (w, p) e, poiché $p \geq p'$, il profitto totale risultante non è minore. Pertanto nessuna soluzione ottima va persa.

La ricorrenza di L_j

Per ogni $j \in \{0, 1, \ldots, n\}$, sia L_j la lista degli stati non dominati che si possono ottenere selezionando un sottoinsieme degli oggetti $\{1, 2, \ldots, j\}$, ordinata per peso strettamente crescente.

La formula ricorsiva per calcolare L_j è la seguente:

$$L_j = \begin{cases} \{(0,0)\} & \text{se } j = 0, \\ \mathrm{nd}\big(L_{j-1} \cup M_j\big) & \text{se } 1 \leq j \leq n, \end{cases} \tag{3}$$

dove $\mathrm{nd}(S)$ indica la lista degli stati non dominati di un insieme S di stati, e

$$M_j = \big\{\, (w + w_j,\ p + p_j) \ :\ (w, p) \in L_{j-1} \,\big\}.$$

Lettura della ricorrenza di L_j (1/2)

- Il processo di decomposizione ricorsiva della formula (3) genera 2 **sottoproblemi indipendenti** per ogni stato di L_{j-1}, corrispondenti alla scelta di selezionare o no l'oggetto j.
- Se l'oggetto j non è selezionato, lo stato non cambia e appartiene a L_{j-1}. Se l'oggetto j è selezionato, lo stato diventa $(w + w_j, p + p_j)$ e appartiene a M_j.
- Qui $L_0 = \{(0, 0)\}$ perché l'unico sottoinsieme di un insieme vuoto di oggetti è il sottoinsieme vuoto.
- Come nella formula (1), la capacità dello zaino non compare nella ricorsione: la lista L_j raccoglie gli stati di *tutti* i sottoinsiemi degli oggetti $\{1, 2, \ldots, j\}$, compresi quelli il cui peso totale supera la capacità.

Lettura della ricorrenza di L_j (2/2)

- Quindi ogni stato non dominato raggiungibile con gli oggetti $\{1, 2, \ldots, j\}$ si costruisce combinando la decisione scelta sull'oggetto j con uno stato non dominato raggiungibile con gli oggetti $\{1, 2, \ldots, j-1\}$.

$$\rightsquigarrow \textbf{Sottostruttura ottima verificata!}$$

- Sottoinsiemi diversi degli oggetti $\{1, 2, \ldots, j\}$ possono dare lo stesso stato, e uno stato può anche essere dominato da un altro. In entrambi i casi in L_j si tiene un solo stato, e tutti i sottoinsiemi che portano a esso vengono trattati insieme nelle iterazioni successive.

$$\rightsquigarrow \textbf{Sottoproblemi sovrapposti verificati!}$$

Il valore ottimo dall'ultima lista

La capacità dello zaino si usa solo alla fine, esattamente come nella formula (2): fra gli stati di L_n, solo quelli il cui peso totale non supera la capacità corrispondono a una soluzione ammissibile.

Il valore ottimo dell'istanza del KP è

$$\max\left\{ p \ : \ (w,p) \in L_n, \ w \leq c \right\}. \tag{4}$$

- L'insieme della formula (4) non è mai vuoto, poiché $(0,0) \in L_n$ e $0 \leq c$, e quindi il massimo è sempre ben definito.

Le liste e le righe della matrice u

La lista L_j contiene in forma compressa l'informazione della riga j della matrice u:

$$\min\left\{w \ : \ (w,p) \in L_j, \ p \geq q\right\} = \min_{q' \geq q} u[j][q'] \quad \forall q \in \{0,1,\ldots,P\}.$$

- I profitti obiettivo con valore $+\infty$ nella riga non lasciano alcuno stato nella lista.
- E nemmeno un profitto obiettivo il cui peso minimo non è minore di quello di un profitto obiettivo più grande.

Esempio: le liste di stati (1/3)

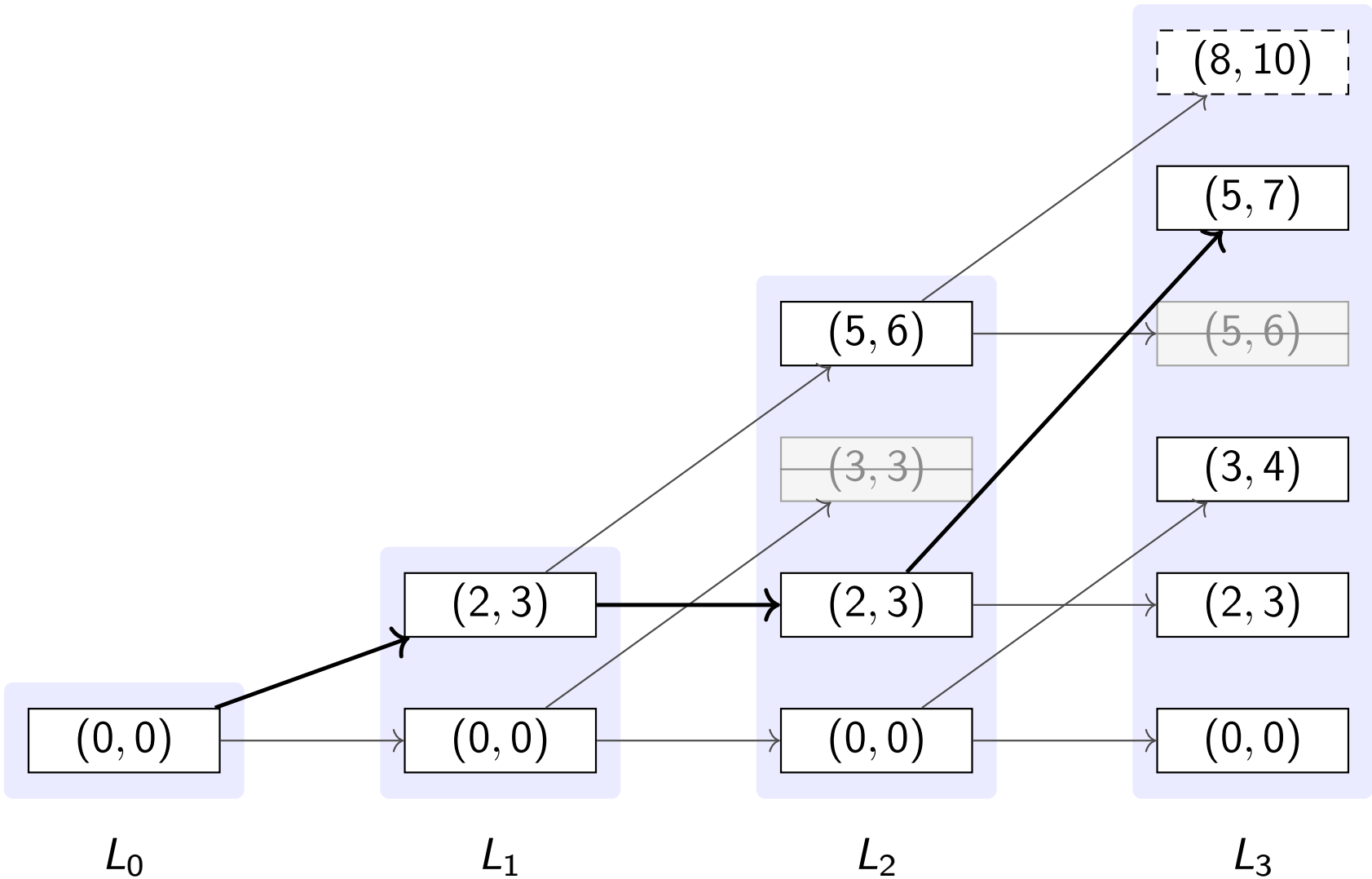
Consideriamo la seconda istanza, con uno zaino di capacità $c = 5$ e i seguenti profitti e pesi:

	$j = 1$	$j = 2$	$j = 3$
p_j	3	3	4
w_j	2	3	3

Nella figura della slide successiva:

- ogni fascia ombreggiata raccoglie gli stati di una lista;
- un arco che entra nella fascia di L_j corrisponde alla decisione presa sull'oggetto j: un arco orizzontale indica che l'oggetto j non è selezionato, mentre un arco obliquo indica che l'oggetto j è selezionato;
- gli archi spessi danno gli stati di una soluzione ottima.

Esempio: le liste di stati (2/3)



Esempio: le liste di stati (3/3)

- I due stati barrati sono dominati e quindi vengono scartati: $(3, 3)$ è dominato da $(2, 3)$ e $(5, 6)$ è dominato da $(5, 7)$.
- Lo stato $(8, 10)$, con il bordo tratteggiato, ha un peso totale maggiore della capacità $c = 5$ dello zaino.
- Resta nella lista, perché la capacità non si usa durante la ricorsione, e viene scartato solo dalla formula (4).

Il calcolo delle liste

- La lista L_{j-1} e la lista M_j sono entrambe ordinate per peso crescente, perché M_j si ottiene da L_{j-1} aggiungendo lo stesso peso w_j a ogni stato.
- Pertanto l'unione della formula (3) si calcola scorrendo contemporaneamente le due liste, come nella fusione di due liste ordinate.

L'ordine di scansione

Per descrivere l'ordine di scansione usiamo la seguente convenzione.

Lo stato (w, p) **precede** lo stato (w', p') se

$$w < w' \qquad \text{o} \qquad (w = w' \text{ e } p > p').$$

- Se gli stati vengono scanditi in quest'ordine, uno stato è non dominato se e solo se il suo profitto è strettamente maggiore del profitto più grande degli stati già scanditi.
- Infatti, ogni stato già scandito ha un peso non maggiore, e quindi domina lo stato corrente non appena il suo profitto non è minore.

Questo è lo pseudocodice dell'**algoritmo di PD con liste**:

Algorithm: DP2_Lists

Input : Un'istanza del KP data dal numero n di oggetti, dai vettori $p[1..n]$ e $w[1..n]$ dei profitti e dei pesi degli oggetti e dalla capacità c dello zaino

Output : Il valore ottimo dell'istanza del KP

```
1 function DP2_Lists(n, p, w, c)
2 |   L := new list containing the single state (0, 0)
3 |   for j := 1 to n do
4 |   |   M := new empty list
5 |   |   for h := 1 to |L| do
6 |   |   |   accodare (L[h].w + w[j], L[h].p + p[j]) a M
7 |   |   end
8 |   |   L ← Merge(L, M)
9 |   end
```

Algorithm: DP2_Lists (continua)

```
10      z := 0
11      for h := 1 to |L| do
12          if L[h].w ≤ c then
13              z ← L[h].p
14          end
15      end
16      return z
17  end
```

Questo è lo pseudocodice della procedura che fonde due liste e scarta gli stati dominati:

Algorithm: Merge

Input : Due liste L e M di stati, entrambe ordinate per peso crescente

Output : La lista degli stati non dominati di $L \cup M$, ordinata per peso crescente

```
1 function Merge(L, M)
2     R := new empty list
3     h := 1
4     k := 1
5     π := −1
```

Algorithm: Merge (continua)

```
6     while h ≤ |L| o k ≤ |M| do
7         if k > |M| o (h ≤ |L| e L[h] precede M[k]) then
8             (a, b) ← L[h]
9             h ← h + 1
10        else
11            (a, b) ← M[k]
12            k ← k + 1
13        end
14        if b > π then
15            accodare (a, b) a R
16            π ← b
17        end
18    end
19    return R
20 end
```

Come funziona DP2_Lists (1/2)

- La lista L è la **lookup table**: al termine dell'iterazione per l'oggetto j, memorizza la lista L_j.
- La lista L si inizializza con il solo stato $(0, 0)$, cioè con la lista L_0.
- Il ciclo esterno, sugli oggetti $j \in \{1, 2, \ldots, n\}$, calcola una lista per ogni oggetto.
- Il ciclo interno costruisce la lista M degli stati ottenuti selezionando l'oggetto j. Qui nessuno stato viene scartato, perché la capacità dello zaino non si usa durante la ricorsione.

Come funziona DP2_Lists (2/2)

- La procedura `Merge` scorre contemporaneamente le due liste e tiene uno stato solo quando il suo profitto è strettamente maggiore del profitto più grande già accodato, memorizzato nella variabile ausiliaria π. Il valore iniziale $\pi = -1$ garantisce che il primo stato scandito venga sempre accodato.
- L'ultimo ciclo realizza la formula (4). Gli stati di L hanno pesi strettamente crescenti e profitti strettamente crescenti, e quindi la variabile ausiliaria z termina con il profitto dell'ultimo stato il cui peso non supera la capacità dello zaino.
- Infine, l'algoritmo restituisce il valore ottimo dell'istanza del KP, z.
- Dopo l'esecuzione dell'algoritmo di PD, la lista L contiene gli stati non dominati L_n, ordinati per peso strettamente crescente e profitto strettamente crescente.

Esempio: esecuzione dell'algoritmo di PD con liste (1/3)

Consideriamo la seconda istanza, con uno zaino di capacità $c = 5$ e i seguenti profitti e pesi:

	$j = 1$	$j = 2$	$j = 3$
p_j	3	3	4
w_j	2	3	3

Per ogni oggetto j, la tabella riporta la lista M_j degli stati ottenuti selezionando l'oggetto j e la lista L_j risultante.

j	M_j	L_j
0	–	$\{(0,0)\}$
1	$\{(2,3)\}$	$\{(0,0),(2,3)\}$
2	$\{(3,3),(5,6)\}$	$\{(0,0),(2,3),(5,6)\}$
3	$\{(3,4),(5,7),(8,10)\}$	$\{(0,0),(2,3),(3,4),(5,7),(8,10)\}$

Esempio: esecuzione dell'algoritmo di PD con liste (2/3)

Per ogni stato scandito da `Merge`, la tabella riporta il valore di j, lo stato, la lista da cui è preso e l'operazione eseguita.

j	stato	da	Azione
1	$(0,0)$	L_0	$0 > \pi = -1$: accodare
1	$(2,3)$	M_1	$3 > \pi = 0$: accodare
2	$(0,0)$	L_1	$0 > \pi = -1$: accodare
2	$(2,3)$	L_1	$3 > \pi = 0$: accodare
2	$(3,3)$	M_2	$3 \leq \pi = 3$: scartare (dominato)
2	$(5,6)$	M_2	$6 > \pi = 3$: accodare

Esempio: esecuzione dell'algoritmo di PD con liste (3/3)

j	stato	da	Azione
3	$(0,0)$	L_2	$0 > \pi = -1$: accodare
3	$(2,3)$	L_2	$3 > \pi = 0$: accodare
3	$(3,4)$	M_3	$4 > \pi = 3$: accodare
3	$(5,7)$	M_3	$7 > \pi = 4$: accodare
3	$(5,6)$	L_2	$6 \leq \pi = 7$: scartare (dominato)
3	$(8,10)$	M_3	$10 > \pi = 7$: accodare

L'ultimo ciclo scorre L_3 e tiene il profitto dell'ultimo stato il cui peso non supera $c = 5$: gli stati $(0,0)$, $(2,3)$, $(3,4)$ e $(5,7)$ danno $z \leftarrow 0$, $z \leftarrow 3$, $z \leftarrow 4$ e $z \leftarrow 7$, mentre lo stato $(8,10)$ non causa alcun aggiornamento poiché $8 > 5$. Pertanto il valore ottimo dell'istanza del KP è 7.

Un'implementazione con liste

Passo 2: calcolo di una soluzione ottima

Le informazioni aggiuntive per le liste (1/2)

- Per calcolare una soluzione ottima, cioè il vettore binario $(x_1, x_2, \ldots, x_n)$, è necessario memorizzare alcune informazioni aggiuntive.
- La sola lista L_n dà il valore ottimo, ma non dice quali oggetti producono lo stato che lo raggiunge.
- Per ogni oggetto j e ogni posizione $h \in \{1, 2, \ldots, |L_j|\}$, oltre allo stato $L_j[h]$, memorizziamo due valori:
  - $\sigma_j[h]$, che è uguale a 1 se l'oggetto j è selezionato nello stato $L_j[h]$, ed è uguale a 0 altrimenti;
  - $\rho_j[h]$, che è la posizione in L_{j-1} dello stato da cui si ottiene $L_j[h]$.

Le informazioni aggiuntive per le liste (2/2)

- Uno stato preso da L_{j-1} dà $\sigma_j[h] = 0$, mentre uno stato preso da M_j dà $\sigma_j[h] = 1$.
- In entrambi i casi $\rho_j[h]$ è la posizione dello stato di L_{j-1} che lo ha generato.
- L'algoritmo di PD deve anche conservare tutte le liste $L_0, L_1, \ldots, L_n$, e non solo l'ultima, perché la ricostruzione le ripercorre all'indietro.

Esempio: calcolo delle informazioni aggiuntive (1/2)

Consideriamo la seconda istanza. Per ogni oggetto j e ogni posizione h, la tabella riporta lo stato $L_j[h]$ e i valori memorizzati $\sigma_j[h]$ e $\rho_j[h]$.

j	h	$L_j[h]$	$\sigma_j[h]$	$\rho_j[h]$
1	1	$(0,0)$	0	1
1	2	$(2,3)$	1	1
2	1	$(0,0)$	0	1
2	2	$(2,3)$	0	2
2	3	$(5,6)$	1	2

Esempio: calcolo delle informazioni aggiuntive (2/2)

j	h	$L_j[h]$	$\sigma_j[h]$	$\rho_j[h]$
3	1	$(0,0)$	0	1
3	2	$(2,3)$	0	2
3	3	$(3,4)$	1	1
3	4	$(5,7)$	1	2
3	5	$(8,10)$	1	3

Per esempio, lo stato $L_3[4] = (5,7)$ ha $\sigma_3[4] = 1$ e $\rho_3[4] = 2$: si ottiene selezionando l'oggetto 3 a partire dallo stato $L_2[2] = (2,3)$, poiché $2 + w_3 = 2 + 3 = 5$ e $3 + p_3 = 3 + 4 = 7$.

Ricostruzione con le liste

- La ricostruzione parte dalla posizione $\tilde{h}$ dello stato che raggiunge il massimo della formula (4), cioè la posizione in cui l'ultimo ciclo di `DP2_Lists` aggiorna z per l'ultima volta.
- In generale non è l'ultima posizione di L_n, perché gli ultimi stati di L_n possono avere un peso totale maggiore della capacità dello zaino.
- Si ricostruisce poi una soluzione ottima scorrendo gli oggetti da $j = n$ fino a $j = 1$, partendo da $h = \tilde{h}$:

$$x_j \leftarrow \sigma_j[h], \quad h \leftarrow \rho_j[h] \quad \text{per } j = n, n-1, \ldots, 1.$$

- Si ottiene così il vettore $(x_1, x_2, \ldots, x_n)$, dove x_j è uguale a 1 se l'oggetto j è selezionato in una soluzione ottima, e a 0 altrimenti.

Algorithm: DP2_Lists_Reconstruct

Input : Il numero n di oggetti, la posizione $\tilde{h}$ dello stato ottimo in L_n e i valori memorizzati σ e ρ

Output : Una soluzione ottima dell'istanza del KP

```
1 function DP2_Lists_Reconstruct(n, h̃, σ, ρ)
2 |   x := new array[1..n]
3 |   for j := 1 to n do
4 |   |   x[j] ← 0
5 |   end
6 |   h := h̃
```

Algorithm: DP2_Lists_Reconstruct (continua)

```
7     for j := n downto 1 do
8         x[j] ← σ_j[h]
9         h ← ρ_j[h]
10    end
11    return x
12 end
```

Esempio: calcolo di una soluzione ottima con le liste (1/2)

Dopo l'esecuzione dell'algoritmo di PD sulla seconda istanza, l'ultima lista è

$$L_3 = \{(0,0),(2,3),(3,4),(5,7),(8,10)\}$$

e il massimo della formula (4) è raggiunto dallo stato $L_3[4] = (5,7)$, quindi $\tilde{h} = 4$ e il valore ottimo è 7.

Esempio: calcolo di una soluzione ottima con le liste (2/2)

Possiamo ricostruire una soluzione ottima partendo da $j = 3$ e $h = \tilde{h} = 4$ come segue:

Iterazione	j	h	$L_j[h]$	$\sigma_j[h]$	$\rho_j[h]$	$x[1]$	$x[2]$	$x[3]$	Azione
–	–	–	–	–	–	0	0	0	$h = 4$
1	3	4	$(5, 7)$	1	2	0	0	1	$h \leftarrow 2$
2	2	2	$(2, 3)$	0	2	0	0	1	$h \leftarrow 2$
3	1	2	$(2, 3)$	1	1	1	0	1	$h \leftarrow 1$

Quindi il vettore

$$(1, 0, 1)$$

è una soluzione ottima. Il valore ottimo corrispondente è $3 + 4 = 7$.

Numero di passi di DP2_Lists

- Indichiamo con $S(n, P)$ il numero di passi eseguiti da `DP2_Lists` nel caso peggiore, dove n è il numero di oggetti e $P = \sum_{j=1}^{n} p_j$ è il profitto totale di tutti gli oggetti.
- Gli stati di una lista hanno profitti strettamente crescenti e ogni profitto appartiene a $\{0, 1, \ldots, P\}$: pertanto $|L_j| \leq P + 1$ per ogni $j \in \{0, 1, \ldots, n\}$, e $|M_j| = |L_{j-1}|$.
- Per ogni oggetto, `DP2_Lists` scorre una volta L_{j-1} per costruire M_j e poi chiama `Merge`, che scandisce $|L_{j-1}| + |M_j| \leq 2(P + 1)$ stati ed esegue un numero costante di operazioni per ciascuno di essi. Il ciclo finale scorre una volta L_n. Pertanto

$$S(n, P) = \Theta(nP).$$

Liste e matrice

- La matrice u ha sempre $(n+1)(P+1)$ celle, la maggior parte delle quali uguali a $+\infty$, mentre le liste contengono solo gli stati effettivamente raggiungibili.
- Sulla seconda istanza le liste contengono 1, 2, 3 e 5 stati, contro le 44 celle della matrice u.
- La lista L_j soddisfa anche $|L_j| \leq 2^j$, poiché gli oggetti $\{1, 2, \ldots, j\}$ hanno 2^j sottoinsiemi. Pertanto

$$|L_j| \leq \min\left\{2^j,\ P+1\right\}.$$

Riepilogo (1/2)

La ricorrenza $u(j, q)$:

- $u(j, q)$ è il minimo peso totale di un sottoinsieme di $\{1, 2, \ldots, j\}$ con profitto totale esattamente q
- $u(0, q) = +\infty$ per $q \geq 1$ esclude i casi inammissibili
- Il valore ottimo dell'istanza del KP è il più grande q con $u(n, q) \leq c$

Il calcolo del valore ottimo:

- Top-down, bottom-up con la matrice $u[0..n][0..P]$, oppure con un solo vettore scorso con profitti obiettivo decrescenti
- $\Theta(nP)$ passi, dove $P = \sum_{j=1}^{n} p_j$

Riepilogo (2/2)

Una soluzione ottima:

- La matrice s si legge all'indietro a partire da $j = n$ e $q = z$

Liste di stati:

- La capacità non si usa durante la ricorsione: gli stati di L_n con $w > c$ si scartano solo alla fine
- La ricostruzione parte dalla posizione $\tilde{h}$ dello stato ottimo, non sempre l'ultima
- $\Theta(nP)$ passi e $|L_j| \leq \min\{2^j, P + 1\}$

Errori tipici

Introduzione: errori tipici

- Elenchiamo gli errori che si commettono più spesso con gli algoritmi di PD per il KP
- Riguardano l'ordine degli aggiornamenti, i casi base, la ricostruzione di una soluzione, il numero di passi e il significato di $u(n, q)$
- Conoscerli aiuta a calcolare correttamente le tabelle a mano

Osservazione 1: Errori tipici (1/2)

- ***Aggiornare il vettore unico in ordine crescente di capacità.*** *La cella $v[k - w_j]$ conterrebbe già la riga j e l'oggetto j potrebbe essere selezionato più di una volta; le capacità si scorrono a partire da c verso il basso.*
- ***Dimenticare i casi base.*** *Senza oggetti disponibili il valore ottimo è 0 per ogni capacità: $v(0, k) = 0$ per ogni $k \in \{0, 1, \ldots, c\}$, non solo per $k = 0$.*
- ***Ricostruire con l'indice sbagliato.*** *A ogni passo l'indice dell'oggetto diminuisce di uno, mentre la capacità diminuisce di w_j solo se l'oggetto j è selezionato.*

Osservazione: Errori tipici (2/2)

- ***Chiamare polinomiale*** $\Theta(nc)$. *La capacità c si scrive con $\log c$ cifre, quindi il tempo di esecuzione è esponenziale nella dimensione dell'input.*
- ***Leggere*** $u(n, q)$ ***come il valore ottimo.*** *Nel secondo algoritmo di PD $u(n, q)$ è un peso minimo; il valore ottimo è il più grande q con $u(n, q) \leq c$.*

Riepilogo

Errori tipici:

- Le capacità del vettore unico si scorrono a partire da c verso il basso
- $v(0, k) = 0$ per ogni capacità k, non solo per $k = 0$
- Nella ricostruzione j diminuisce di uno, k di w_j solo se l'oggetto j è selezionato
- $\Theta(nc)$ è pseudopolinomiale, non polinomiale
- $u(n, q)$ è un peso minimo, non il valore ottimo

In sintesi

Introduzione: in sintesi

- Raccogliamo le idee chiave del capitolo
- Chiudiamo con il cruscotto della prima istanza: i bound trovati finora

Riepilogo (1/2)

- La ricorrenza $v(j, k)$ sceglie fra lasciare fuori l'oggetto j e prenderlo; ha la sottostruttura ottima e i sottoproblemi sovrapposti.
- La matrice si riempie bottom-up in $\Theta(nc)$ passi; un solo vettore scorso verso il basso dà il valore ottimo con sole $c + 1$ celle.
- Una soluzione ottima si ricava dalla matrice s delle decisioni, letta all'indietro a partire da (n, c).
- Il tempo di esecuzione $\Theta(nc)$ è pseudopolinomiale; le liste di stati non dominati memorizzano solo gli stati raggiungibili.
- Scambiando profitti e pesi si ottengono $\Theta(nP)$ passi, il che è meglio quando il profitto totale P è piccolo.

Riepilogo (2/2)

Cruscotto dell'esempio del KP. La prima istanza ($c = 5$, tre oggetti) con i bound trovati finora:

Metodo	Lower bound	Upper bound	Origine
Algoritmo di PD	14	14	valore ottimo, $\Theta(nc)$ passi