PD per il problema dello zaino con setup

Indice

Un algoritmo di PD
    Numero di passi

Errori tipici

In sintesi

Obiettivi

Al termine di questo capitolo saremo in grado di:

- scrivere la ricorrenza del KPS: per ogni classe, il valore migliore senza aprirla e il valore migliore dopo averne pagato il setup;
- eseguire a mano l'algoritmo di PD, classe per classe, con una ricorrenza del KP dentro ogni classe aperta;
- ricostruire una soluzione ottima dalle tabelle delle scelte s e o;
- contare i $\Theta(nc)$ passi dell'algoritmo di PD.

Prerequisiti

- Da questo modulo, il capitolo «PD per il problema dello zaino 0/1»: la ricorrenza $v(j, k)$ del problema dello zaino 0/1 (KP) su oggetti e capacità, la matrice s delle decisioni e la ricostruzione di una soluzione.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Problemi di zaino»: il problema dello zaino con setup, le sue classi, i costi di setup f_h e i pesi di setup s_h.
- Dal modulo *Algoritmi*: le convenzioni dello pseudocodice e la dimensione dell'input.

Il problema di questo capitolo

Il problema di questo capitolo è il problema dello zaino con setup (KPS): la sua definizione e le sue ipotesi sono nella sua pagina.

Un algoritmo di PD

Introduzione: Un algoritmo di PD

- Costruiamo un algoritmo di PD per il problema dello zaino con setup (KPS)
- Le classi sono gli stadi: ogni classe resta chiusa, oppure viene aperta pagandone il setup una volta sola
- Dentro una classe aperta, la scelta degli oggetti è un problema dello zaino 0/1 (KP): riusiamo l'algoritmo di PD del KP
- Calcoliamo il valore ottimo, poi una soluzione ottima da due tabelle delle scelte, e infine contiamo il numero di passi

Oggetti, classi e soluzioni

- Numeriamo gli oggetti classe per classe, in modo che

$$\mathscr{J}_h = \{\ell_{h-1} + 1, \ldots, \ell_h\}$$

per ogni classe h, con $\ell_0 = 0$ e $\ell_q = n$.

- La classe h ha $t_h = |\mathscr{J}_h| = \ell_h - \ell_{h-1}$ oggetti, e il suo i-esimo oggetto è $j = \ell_{h-1} + i$.

- Una soluzione si descrive con il vettore binario $(x_1, x_2, \ldots, x_n)$.

- La classe h è **aperta** se almeno uno dei suoi oggetti è selezionato: allora il suo peso di setup s_h si aggiunge al peso totale, e il suo costo di setup f_h si sottrae dal ricavo totale.

Esempio: istanza del KPS

Consideriamo uno zaino di capacità $c = 10$ e un insieme $\mathscr{I} = \{1, 2, \ldots, 5\}$ di 5 oggetti ($n = 5$), partizionato in $q = 2$ classi:

- $\mathscr{J}_1 = \{1, 2\}$, con costo di setup $f_1 = 2$ e peso di setup $s_1 = 2$;
- $\mathscr{J}_2 = \{3, 4, 5\}$, con costo di setup $f_2 = 3$ e peso di setup $s_2 = 1$.

I ricavi e i pesi sono:

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
p_j	6	5	4	7	3
w_j	3	2	2	4	1

Useremo questa istanza del KPS in tutto il capitolo.

Passo 1: calcolo del valore ottimo

- Le classi sono gli stadi dell'algoritmo di PD.
- Per ogni classe h ci sono due opzioni:
  - la classe h resta chiusa, oppure
  - la classe h viene aperta, se ne paga il setup e si selezionano alcuni dei suoi oggetti.
- Dentro una classe aperta, la scelta degli oggetti è un KP: l'algoritmo di PD del KP scorre gli oggetti della classe, partendo dal valore rimasto dopo aver pagato il setup.

I valori $v(h,k)$ e $u(h,i,k)$

- Per ogni $h \in \{0,1,\ldots,q\}$ e $k \in \{0,1,\ldots,c\}$, sia $v(h,k)$ il valore ottimo della sottoistanza del KPS con capacità k e le classi $\mathscr{J}_1,\ldots,\mathscr{J}_h$.

- Per ogni $h \in \{1,\ldots,q\}$ e $i \in \{0,1,\ldots,t_h\}$, sia $u(h,i,k)$ il valore ottimo della stessa sottoistanza quando la classe h è aperta (il suo setup è pagato) e sono disponibili solo i suoi primi i oggetti, con $u(h,i,k) = -\infty$ se il peso di setup non entra nello zaino.

Ora: le formule ricorsive per calcolare $u(h,i,k)$ e $v(h,k)$.

La ricorrenza per $u(h, i, k)$

La formula ricorsiva per calcolare $u(h, i, k)$ è la seguente, dove $j = \ell_{h-1} + i$ è l'i-esimo oggetto della classe h:

$$u(h,i,k) = \begin{cases} -\infty & \text{se } i = 0 \text{ e } k < s_h, \\ v(h-1, k-s_h) - f_h & \text{se } i = 0 \text{ e } k \geq s_h, \\ u(h, i-1, k) & \text{se } i \geq 1 \text{ e } w_j > k, \\ \max \left\{ u(h, i-1, k), \right. & \text{se } i \geq 1 \text{ e } w_j \leq k. \\ \qquad \left. p_j + u(h, i-1, k-w_j) \right\} & \end{cases} \tag{1}$$

La ricorrenza per $v(h, k)$

La formula ricorsiva per calcolare $v(h, k)$ è la seguente:

$$v(h,k) = \begin{cases} 0 & \text{se } h = 0, \\ \max\left\{v(h-1,k),\ u(h,t_h,k)\right\} & \text{se } h \geq 1. \end{cases} \tag{2}$$

Come funzionano le ricorrenze

- La riga $u(h, 0, \cdot)$ **paga il setup** della classe h: il peso di setup s_h lascia la capacità $k - s_h$ alle classi $1, 2, \ldots, h-1$, e il costo di setup f_h viene sottratto.
- Le righe $u(h, 1, \cdot), \ldots, u(h, t_h, \cdot)$ sono la ricorrenza del KP sugli oggetti della classe h, con $-\infty + p_j = -\infty$.
- La formula (2) confronta le due opzioni per la classe h: chiusa, con valore $v(h-1, k)$, oppure aperta, con valore $u(h, t_h, k)$.

Una classe aperta senza oggetti selezionati

- La riga $u(h, 0, \cdot)$ ammette una classe aperta senza oggetti selezionati, che non è una soluzione del KPS.
- Questo valore non viene mai scelto in (2), perché $f_h \geq 1$ dà

$$v(h-1, k-s_h) - f_h \; < \; v(h-1, k).$$

- Quindi, quando il massimo in (2) è raggiunto solo da $u(h, t_h, k)$, almeno un oggetto della classe h è selezionato.

Sottostruttura ottima e sottoproblemi sovrapposti

- Una soluzione ottima per (h, k) si costruisce da una soluzione ottima di una sottoistanza più piccola: $(h-1, k)$ se la classe h è chiusa e, dentro una classe aperta, $(h, i-1, \cdot)$ con o senza l'oggetto j.

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

- Decisioni diverse sulle classi $h+1, \ldots, q$ possono lasciare la stessa capacità k, e quindi lo stesso valore $v(h, k)$ può servire più volte.

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

L'approccio bottom-up

- L'**approccio bottom-up** calcola, per $h = 1, 2, \ldots, q$, prima le righe $u(h, i, \cdot)$ per $i = 0, 1, \ldots, t_h$, e poi la riga $v(h, \cdot)$, ciascuna per k crescente.

$$
\begin{aligned}
& v(0, \cdot) \\
h = 1: \quad & u(1, 0, \cdot),\ u(1, 1, \cdot),\ \ldots,\ u(1, t_1, \cdot) \ \rightarrow\ v(1, \cdot) \\
& \vdots \\
h = q: \quad & u(q, 0, \cdot),\ u(q, 1, \cdot),\ \ldots,\ u(q, t_q, \cdot) \ \rightarrow\ v(q, \cdot)
\end{aligned}
$$

- Il valore ottimo dell'istanza del KPS è $v(q, c)$.

Passo 2: calcolo di una soluzione ottima

- Si memorizzano due tabelle delle scelte:
  - $s[h][i][k] = 1$ se l'i-esimo oggetto della classe h è selezionato nel valore ottimo $u[h][i][k]$;
  - $o[h][k] = 1$ se la classe h è aperta nel valore ottimo $v[h][k]$.
- Il peso di setup s_h ha un indice; la scelta $s[h][i][k]$ ne ha tre.

Ora: lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**,
che memorizza le tabelle delle scelte insieme ai valori.

Algorithm: KPS_DP (1/4)

Input : Un'istanza del KPS data da q, dalle classi $\mathscr{J}_1, \ldots, \mathscr{J}_q$ numerate come sopra, dai costi di setup f_h e dai pesi di setup s_h, dai vettori $p[1..n]$, $w[1..n]$ e dalla capacità c

Output: Il valore ottimo dell'istanza del KPS e le tabelle delle scelte s e o

```
1 function KPS_DP(q, J, p, w, c)
2 |   v, o := new arrays[0..q][0..c];
  |    u, s := new arrays[1..q][0..t_h][0..c]
3 |   for k := 0 to c do
4 |   |   v[0][k] ← 0; o[0][k] ← 0
5 |   end
```

Algorithm: KPS_DP (2/4)

function *KPS_DP*$(q, \mathscr{J}, p, w, c)$ (continua)

```
6   |   for h := 1 to q do
7   |   |   for k := 0 to c do
8   |   |   |   u[h][0][k] ← −∞
9   |   |   |   if k ≥ s_h then
10  |   |   |   |   u[h][0][k] ← v[h − 1][k − s_h] − f_h
11  |   |   |   end
12  |   |   end
```

Algorithm: KPS_DP (3/4)

function *KPS_DP*$(q, \mathscr{J}, p, w, c)$ (continua)

```
    for h := 1 to q do (continua)
13      for i := 1 to t_h do
14          j := ℓ_{h-1} + i
15          for k := 0 to c do
16              u[h][i][k] ← u[h][i − 1][k]; s[h][i][k] ← 0
17              if w[j] ≤ k e
                  p[j] + u[h][i − 1][k − w[j]] > u[h][i][k]
                  then
18                    u[h][i][k] ← p[j] + u[h][i − 1][k − w[j]];
                      s[h][i][k] ← 1
19              end
20          end
21      end
```

Algorithm: KPS_DP (4/4)

```
function KPS_DP(q, 𝒥, p, w, c) (continua)
    for h := 1 to q do (continua)
22      for k := 0 to c do
23          v[h][k] ← v[h − 1][k]; o[h][k] ← 0
24          if u[h][t_h][k] > v[h][k] then
25              v[h][k] ← u[h][t_h][k]; o[h][k] ← 1
26          end
27      end
28  end
29  return v[q][c], s, o
30 end
```

Come funziona KPS_DP

- Per ogni classe:
  - il primo ciclo su k paga il setup dove entra nello zaino;
  - i cicli su i e k eseguono l'algoritmo di PD del KP sugli oggetti della classe;
  - l'ultimo ciclo su k confronta la classe chiusa con quella aperta.
- In caso di pareggio l'oggetto non viene selezionato e la classe resta chiusa, perché i valori vengono aggiornati solo quando sono strettamente maggiori.

Esempio: esecuzione dell'algoritmo di PD (1/6)

Consideriamo l'istanza del KPS dell'esempio: $c = 10$, $\mathscr{J}_1 = \{1, 2\}$ con $f_1 = 2$ e $s_1 = 2$, $\mathscr{J}_2 = \{3, 4, 5\}$ con $f_2 = 3$ e $s_2 = 1$, e

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
p_j	6	5	4	7	3
w_j	3	2	2	4	1

Le righe si calcolano nell'ordine della tabella dei valori, ciascuna per $k = 0, 1, \ldots, 10$.

Esempio: esecuzione dell'algoritmo di PD (2/6)

Questi sono alcuni dei calcoli:

- $u[1][0][1] = -\infty$, perché $k = 1 < s_1 = 2$
- $v[1][7] = \max\{v[0][7], u[1][2][7]\} = \max\{0, 9\} = 9$
- $v[2][10] = \max\{v[1][10], u[2][3][10]\} = \max\{9, 11\} = 11$
- $u[2][3][10] = \max\{u[2][2][10], p_5 + u[2][2][9]\} = \max\{10, 3 + 8\} = 11$

Esempio: esecuzione dell'algoritmo di PD (3/6)

Questi sono alcuni dei calcoli (continua):

- $u[2][2][9] = \max\{u[2][1][9],\ p_4 + u[2][1][5]\} = \max\{6,\ 7 + 1\} = 8$
- $u[2][1][5] = \max\{u[2][0][5],\ p_3 + u[2][0][3]\} = \max\{0,\ 4 - 3\} = 1$
- $u[2][0][3] = v[1][2] - f_2 = 0 - 3 = -3$
- $v[1][2] = \max\{v[0][2],\ u[1][2][2]\} = \max\{0,\ -2\} = 0$

Esempio: esecuzione dell'algoritmo di PD (4/6)

La tabella dei valori dopo l'esecuzione dell'algoritmo di PD, riga per riga nell'ordine di calcolo ($k = 0, 1, \ldots, 5$):

Riga	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$v[0][k]$	0	0	0	0	0	0
$u[1][0][k]$ (setup)	$-\infty$	$-\infty$	-2	-2	-2	-2
$u[1][1][k]$ ($j=1$)	$-\infty$	$-\infty$	-2	-2	-2	4
$u[1][2][k]$ ($j=2$)	$-\infty$	$-\infty$	-2	-2	3	4
$v[1][k]$	0	0	0	0	3	4
$u[2][0][k]$ (setup)	$-\infty$	-3	-3	-3	-3	0
$u[2][1][k]$ ($j=3$)	$-\infty$	-3	-3	1	1	1
$u[2][2][k]$ ($j=4$)	$-\infty$	-3	-3	1	1	4
$u[2][3][k]$ ($j=5$)	$-\infty$	-3	0	1	4	4
$v[2][k]$	0	0	0	1	4	4

Esempio: esecuzione dell'algoritmo di PD (5/6)

Riga	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$v[0][k]$	0	0	0	0	0
$u[1][0][k]$ (setup)	-2	-2	-2	-2	-2
$u[1][1][k]$ $(j=1)$	4	4	4	4	4
$u[1][2][k]$ $(j=2)$	4	9	9	9	9
$v[1][k]$	4	9	9	9	9
$u[2][0][k]$ (setup)	1	1	6	6	6
$u[2][1][k]$ $(j=3)$	1	4	6	6	10
$u[2][2][k]$ $(j=4)$	4	8	8	8	10
$u[2][3][k]$ $(j=5)$	7	8	11	11	11
$v[2][k]$	7	9	11	11	11

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

- Il valore ottimo dell'istanza è

$$v[2][10] = 11.$$

- I valori -2 e -3 nelle righe di setup sono i costi di setup pagati prima di selezionare qualsiasi oggetto.

Esempio: le tabelle delle scelte (1/2)

Per l'istanza dell'esempio, queste sono le tabelle delle scelte calcolate dall'algoritmo di PD: le righe $s[h][i][\cdot]$ degli oggetti e le righe $o[h][\cdot]$ delle classi ($k = 0, 1, \ldots, 5$).

Riga	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$s[1][1][k]$ $(j=1)$	0	0	0	0	0	1
$s[1][2][k]$ $(j=2)$	0	0	0	0	1	0
$o[1][k]$	0	0	0	0	1	1
$s[2][1][k]$ $(j=3)$	0	0	0	1	1	1
$s[2][2][k]$ $(j=4)$	0	0	0	0	0	1
$s[2][3][k]$ $(j=5)$	0	0	1	0	1	0
$o[2][k]$	0	0	0	1	1	0

Esempio: le tabelle delle scelte (2/2)

Le tabelle delle scelte ($k = 6, 7, \ldots, 10$):

Riga	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$s[1][1][k]$ $(j=1)$	1	1	1	1	1
$s[1][2][k]$ $(j=2)$	0	1	1	1	1
$o[1][k]$	1	1	1	1	1
$s[2][1][k]$ $(j=3)$	0	1	0	0	1
$s[2][2][k]$ $(j=4)$	1	1	1	1	0
$s[2][3][k]$ $(j=5)$	1	0	1	1	1
$o[2][k]$	1	0	1	1	1

Ricostruzione di una soluzione ottima

- Una soluzione ottima si ricostruisce scorrendo le classi da $h = q$ fino a $h = 1$, partendo da $k = c$ e da $x_j = 0$ per ogni oggetto j.
- Se $o[h][k] = 0$, la classe h è chiusa e k non cambia.
- Se $o[h][k] = 1$, gli oggetti della classe h si scorrono da $i = t_h$ fino a $i = 1$ come nel KP, e alla fine il peso di setup s_h si sottrae da k.

Algorithm: KPS_Reconstruct (1/2)

Input : Il numero q di classi, i pesi di setup s_h, il vettore $w[1..n]$, la capacità c e le tabelle delle scelte s e o

Output: Una soluzione ottima dell'istanza del KPS

```
1 function KPS_Reconstruct(q, w, c, s, o)
2 |   x := new array[1..n], con x[j] = 0 per ogni j
3 |   k := c
```

Algorithm: KPS_Reconstruct (2/2)

function *KPS_Reconstruct*(q, w, c, s, o) (continua)

```
4   |  for h := q downto 1 do
5   |  |  if o[h][k] = 1 then
6   |  |  |  for i := t_h downto 1 do
7   |  |  |  |  j := ℓ_{h-1} + i; x[j] ← s[h][i][k]
8   |  |  |  |  k ← k − x[j] w[j]
9   |  |  |  end
10  |  |  |  k ← k − s_h
11  |  |  end
12  |  end
13  |  return x
14  end
```

Esempio: calcolo di una soluzione ottima (1/4)

Dopo l'esecuzione dell'algoritmo di PD sull'istanza dell'esempio, partiamo da $h = 2$ e $k = 10$. Le scelte lette nelle tabelle:

Iterazione	Stadio	k	Scelta	Azione
–	–	–	–	$k = 10$
1	$h = 2$	10	$o[2][10] = 1$	classe 2 aperta
2	$j = 5$	10	$s[2][3][10] = 1$	$k \leftarrow 10 - 1 = 9$
3	$j = 4$	9	$s[2][2][9] = 1$	$k \leftarrow 9 - 4 = 5$
4	$j = 3$	5	$s[2][1][5] = 1$	$k \leftarrow 5 - 2 = 3$
–	–	3	–	setup: $k \leftarrow 3 - s_2 = 2$
5	$h = 1$	2	$o[1][2] = 0$	classe 1 chiusa

Esempio: calcolo di una soluzione ottima (2/4)

Le stesse iterazioni, con il vettore x:

Iterazione	Stadio	k	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$
–	–	–	0	0	0	0	0
1	$h = 2$	10	0	0	0	0	0
2	$j = 5$	10	0	0	0	0	1
3	$j = 4$	9	0	0	0	1	1
4	$j = 3$	5	0	0	1	1	1
–	–	3	0	0	1	1	1
5	$h = 1$	2	0	0	1	1	1

Esempio: calcolo di una soluzione ottima (3/4)

- Il vettore
$$x = (0, 0, 1, 1, 1)$$
è una soluzione ottima dell'istanza del KPS dell'esempio.
- Solo la classe 2 è aperta, e sono selezionati gli oggetti $\{3, 4, 5\}$.
- Il suo peso totale è $2 + 4 + 1 + s_2 = 8 \leq c$.
- Il suo valore ottimo è $v(2, 10) = 4 + 7 + 3 - f_2 = 11$.

Esempio: calcolo di una soluzione ottima (4/4)

- Gli oggetti della classe 1 hanno il rapporto maggiore fra ricavo totale e peso totale (11/5 contro 14/7), ma la classe 1 resta chiusa.
- Con la sola classe 1 il valore migliore è $v[1][10] = 9$, mentre la classe 2 aperta dà $u[2][3][10] = 11$.
- Dopo i suoi oggetti e il suo peso di setup resta solo $k = 2$ per la classe 1, e $o[1][2] = 0$.

Numero di passi

- Indichiamo con $S(n, q, c)$ il numero di passi eseguiti da `KPS_DP` nel caso peggiore.
- I cicli su i scorrono gli oggetti di ogni classe, quindi su tutte le classi vengono eseguiti $t_1 + t_2 + \cdots + t_q = n$ volte.
- Supponiamo che le condizioni alle righe 9, 17 e 24 siano sempre vere.
- I numeri di riga sono quelli delle quattro parti dello pseudocodice di `KPS_DP`.

Numero di esecuzioni delle righe

- Le righe 2 e 29 vengono eseguite 1 volta ciascuna.
- Le righe 3 e 4 vengono eseguite $c + 1$ volte ciascuna.
- La riga 6 viene eseguita q volte.
- Le righe 7, 8, 9, 10, 22, 23, 24 e 25 vengono eseguite $q(c + 1)$ volte ciascuna.
- Le righe 13 e 14 vengono eseguite n volte ciascuna.
- Le righe 15, 16, 17 e 18 vengono eseguite $n(c + 1)$ volte ciascuna.

Il numero totale di passi

Quindi il numero totale di passi nel caso peggiore è

$$S(n, q, c) = 4n(c + 1) + 8q(c + 1) + 2n + q + 2c + 4.$$

Poiché ogni classe contiene almeno un oggetto, $1 \leq q \leq n$, e

$$4n(c + 1) \leq S(n, q, c) \leq 12n(c + 1) + 3n + 2c + 4.$$

Entrambi i bound sono $\Theta(nc)$, quindi

$$S(n, q, c) = \Theta(nc).$$

Il numero di passi dell'algoritmo di PD

L'algoritmo di PD per il KPS esegue

$$S(n, q, c) = \Theta(nc)$$

passi nel caso peggiore: i setup aggiungono $2q$ righe alle n righe della ricorrenza del KP.

- La ricostruzione esegue $\Theta(n)$ passi.

Riepilogo (1/2)

Le ricorrenze:

- $v(h, k)$: valore ottimo con capacità k e le classi $\mathscr{J}_1, \ldots, \mathscr{J}_h$; $u(h, i, k)$: lo stesso con la classe h aperta e i suoi primi i oggetti disponibili
- $u(h, 0, k) = v(h-1, k-s_h) - f_h$ paga il setup (oppure $-\infty$ se $k < s_h$); le righe $u(h, 1, \cdot), \ldots, u(h, t_h, \cdot)$ sono la ricorrenza del KP sugli oggetti della classe h
- $v(h, k) = \max\{v(h-1, k),\ u(h, t_h, k)\}$: classe chiusa o aperta

L'algoritmo di PD:

- Sottostruttura ottima e sottoproblemi sovrapposti verificati
- Bottom-up, classe per classe; sull'esempio $v[2][10] = 11$

Riepilogo (2/2)

Una soluzione ottima:

- Le tabelle delle scelte $s[h][i][k]$ (oggetti) e $o[h][k]$ (classi)
- Da $h = q$ e $k = c$: se $o[h][k] = 1$, gli oggetti della classe h si scorrono come nel KP e poi s_h si sottrae da k
- Sull'esempio: $x = (0, 0, 1, 1, 1)$, solo la classe 2 aperta

Numero di passi:

- $S(n, q, c) = 4n(c + 1) + 8q(c + 1) + 2n + q + 2c + 4 = \Theta(nc)$; la ricostruzione esegue $\Theta(n)$ passi

Errori tipici

Introduzione: Errori tipici

- Elenchiamo gli errori che si commettono più spesso con l'algoritmo di PD per il KPS
- Riguardano dove si paga il setup, la partenza delle righe di una classe aperta, la ricostruzione di una soluzione e la notazione
- Conoscerli aiuta a eseguire correttamente a mano l'algoritmo di PD

Osservazione 1: Errori tipici (1/2)

- ***Pagare il setup per ogni oggetto.*** *Il setup della classe h si paga una volta sola, nella riga $u(h, 0, \cdot)$; le righe degli oggetti aggiungono solo p_j e w_j.*
- ***Far partire gli oggetti di una classe da*** $v(h-1, k)$**.** *Le righe di una classe aperta partono da $v(h-1, k-s_h) - f_h$: partire da $v(h-1, k)$ selezionerebbe oggetti senza pagare il setup.*

Osservazione: Errori tipici (2/2)

- ***Dimenticare il peso di setup nella ricostruzione.*** *Dopo gli oggetti di una classe aperta, la capacità diminuisce anche di s_h.*
- ***Confondere s_h con $s[h][i][k]$.*** *Il peso di setup della classe h ha un indice; la scelta dell'i-esimo oggetto della classe h ne ha tre.*

Riepilogo

Errori tipici:

- Il setup di una classe si paga una volta sola, nella riga $u(h, 0, \cdot)$, non per ogni oggetto
- Le righe di una classe aperta partono da $v(h-1, k-s_h) - f_h$, non da $v(h-1, k)$
- Nella ricostruzione, la capacità di una classe aperta diminuisce anche di s_h
- Il peso di setup s_h ha un indice, la scelta $s[h][i][k]$ ne ha tre

In sintesi

Introduzione: In sintesi

- Raccogliamo le idee chiave del capitolo sull'algoritmo di PD per il KPS
- Chiudiamo con il quadro riassuntivo dell'esempio del capitolo

Riepilogo (1/2)

- Per ogni classe h l'algoritmo di PD confronta la classe chiusa, $v(h-1,k)$, con la classe aperta, $u(h,t_h,k)$.
- La classe aperta parte da $u(h,0,k) = v(h-1,k-s_h) - f_h$, che paga il setup una volta sola, e prosegue con la ricorrenza del KP sui suoi oggetti.
- Le tabelle delle scelte s (oggetti) e o (classi) danno una soluzione ottima, letta all'indietro a partire da (q,c).
- L'algoritmo di PD esegue $\Theta(nc)$ passi, come l'algoritmo di PD del KP.

Riepilogo (2/2)

Quadro riassuntivo dell'esempio del KPS ($c = 10$, due classi):

Metodo	Lower bound	Upper bound	Origine
Algoritmo di PD	11	11	valore ottimo, soluzione $x = (0, 0, 1, 1, 1)$