PD per il problema dello zaino limitato

Indice

Istanze e soluzioni ammissibili
    Codifica delle soluzioni ammissibili

Un algoritmo di PD
    Numero di passi

Errori tipici

In sintesi

Obiettivi

Al termine di questo capitolo saremo in grado di:

- spiegare perché il KP ($b_j = 1$) e l'UKP ($b_j \geq \lfloor c/w_j \rfloor$) sono casi particolari del BKP, e scriverne la ricorrenza;
- eseguire a mano l'algoritmo di PD, provando ogni numero ammissibile d di copie di ciascun tipo di oggetto;
- riempire la matrice s dei numeri di copie e ricostruire da essa una soluzione ottima;
- contare i passi dell'algoritmo di PD e dire quando $\Theta(nc\, b_{\max})$ è stretto.

Prerequisiti

- Da questo modulo, i capitoli «PD per il problema dello zaino 0/1» e «PD per il problema dello zaino illimitato»: la ricorrenza $v(j, k)$ del problema dello zaino 0/1 (KP) e del problema dello zaino illimitato (UKP) su oggetti e capacità, e la ricostruzione di una soluzione.
- Da questo modulo, il capitolo «Principi della programmazione dinamica»: sottostruttura ottima, sottoproblemi sovrapposti, casi base.
- 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 limitato (BKP).

La sua definizione e le sue ipotesi sono nella sua pagina.

Istanze e soluzioni ammissibili

Introduzione: istanze e soluzioni ammissibili

- Fissiamo l'istanza del BKP che usiamo in tutti gli esempi del capitolo
- Vediamo in che cosa il BKP differisce dal problema dello zaino 0/1 (KP) e dal problema dello zaino illimitato (UKP)
- Contiamo le soluzioni possibili: enumerarle diventa rapidamente impraticabile, quindi serve un metodo migliore
- Descriviamo una soluzione ammissibile con un vettore intero: è ciò che calcolerà l'algoritmo di PD

Esempio: istanza del BKP

Consideriamo uno zaino di capacità $c = 10$ e 3 tipi di oggetto, con i seguenti profitti, pesi e limiti:

	$j = 1$	$j = 2$	$j = 3$
p_j	6	10	12
w_j	2	3	5
b_j	2	2	1

Usiamo questa istanza del BKP in tutti gli esempi del capitolo.

Dal KP al BKP

- La differenza principale rispetto al problema dello zaino 0/1 (KP) è che ogni tipo di oggetto può essere selezionato **più di una volta**
- La variabile x_j non è più binaria, ma è limitata da b_j
- Il BKP si riduce al KP quando $b_j = 1$ per ogni tipo di oggetto j
- Il BKP si riduce al problema dello zaino illimitato (UKP) quando ogni limite b_j è abbastanza grande da non essere vincolante

Quante soluzioni?

- Il numero di vettori possibili $(x_1, x_2, \ldots, x_n)$, con $x_j \in \{0, 1, \ldots, b_j\}$ per ogni tipo di oggetto j, è

$$\prod_{j=1}^{n} (b_j + 1).$$

- Alcuni di questi vettori possono non rispettare la capacità dello zaino
- Quindi il numero di soluzioni ammissibili è al più $\prod_{j=1}^{n} (b_j + 1)$, una quantità che **cresce esponenzialmente** con n
- Enumerare tutte le soluzioni possibili diventa rapidamente impraticabile al crescere di n

Soluzioni ammissibili

Una *soluzione ammissibile* del BKP è una scelta del numero di copie di ciascun tipo di oggetto che rispetta sia i limiti b_j sia la capacità c dello zaino.

- Una soluzione ammissibile del BKP si può descrivere con il vettore intero

$$(x_1, x_2, \ldots, x_n)$$

Codifica delle soluzioni ammissibili

- Per ogni tipo di oggetto $j \in \{1, 2, \ldots, n\}$, ogni componente ha il seguente significato:

$$x_j = \text{numero di copie selezionate del tipo di oggetto } j,$$

$$0 \leq x_j \leq b_j.$$

- Poiché la soluzione è ammissibile, il peso totale degli oggetti selezionati non deve superare la capacità dello zaino:

$$\sum_{j=1}^{n} w_j x_j \leq c.$$

- Il profitto totale di una soluzione ammissibile è

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

Riepilogo

Istanze:

- L'istanza degli esempi: $c = 10$ e tre tipi di oggetto, con limiti $b_1 = 2$, $b_2 = 2$, $b_3 = 1$
- Ogni tipo di oggetto può essere selezionato fino a b_j volte: KP quando $b_j = 1$, UKP quando i limiti non sono vincolanti

Soluzioni ammissibili:

- Al più $\prod_{j=1}^{n}(b_j + 1)$ vettori candidati: un numero esponenziale in n
- Una soluzione ammissibile è un vettore intero $(x_1, x_2, \ldots, x_n)$ con $0 \leq x_j \leq b_j$ e $\sum_{j=1}^{n} w_j x_j \leq c$; il suo profitto è $\sum_{j=1}^{n} p_j x_j$

Un algoritmo di PD

Introduzione: un algoritmo di PD

- **Passo 1**: definiamo i sottoproblemi $v(j, k)$ e una ricorrenza che prova ogni numero ammissibile d di copie del tipo di oggetto j
- Verifichiamo la sottostruttura ottima e i sottoproblemi sovrapposti, e confrontiamo l'approccio top-down e l'approccio bottom-up
- **Passo 2**: memorizziamo il numero di copie scelto in ogni sottoproblema e ricostruiamo una soluzione ottima
- Infine contiamo i passi: l'algoritmo di PD esegue $\Theta(nc\, b_{\max})$ passi, un numero pseudo-polinomiale, mentre l'enumerazione è esponenziale

Passo 1: calcolo del valore ottimo

Per ogni $j \in \{0, 1, \ldots, n\}$ e $k \in \{0, 1, \ldots, c\}$, sia $v(j, k)$ il valore ottimo della sottoistanza del BKP data da:

1. uno zaino di capacità k,
2. un insieme $\{1, 2, \ldots, j\}$ di tipi di oggetto disponibili,
3. per ogni tipo di oggetto $h \in \{1, 2, \ldots, j\}$, un profitto p_h, un peso w_h e un limite b_h.

La ricorrenza

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

$$v(j,k) = \begin{cases} 0 & \text{se } j = 0 \text{ o } k = 0, \\ \max\limits_{d \in \{0,1,\ldots,\min\{b_j, \lfloor k/w_j \rfloor\}\}} \left\{ d\, p_j + v(j-1, k - d\, w_j) \right\} & \\ & \text{se } 1 \leq j \leq n \text{ e } 1 \leq k \leq c. \end{cases}$$

Sapienza Università di Roma

Sottoproblemi e casi base

- Il processo di decomposizione ricorsiva della ricorrenza genera al più $b_j + 1$ **sottoproblemi indipendenti**
- Corrispondono ai possibili numeri $d \in \{0, 1, \ldots, \min\{b_j, \lfloor k/w_j \rfloor\}\}$ di copie del tipo di oggetto j inserite in uno zaino di capacità k, usando solo i tipi di oggetto $1, 2, \ldots, j$
- I **casi base** sono:
  - $v(0, k) = 0$, perché nessun tipo di oggetto è disponibile;
  - $v(j, 0) = 0$, perché non c'è capacità disponibile.

Sottostruttura ottima (1/2)

- Il valore ottimo $v(j, k)$ si ottiene provando ogni numero ammissibile d di copie del tipo di oggetto j e scegliendo quello che massimizza la somma

$$d\, p_j + v(j-1, k - d\, w_j)$$

del profitto delle d copie e del valore ottimo del sottoproblema residuo, ristretto ai tipi di oggetto $1, 2, \ldots, j-1$

Sottostruttura ottima (2/2)

- Quindi una soluzione ottima della sottoistanza con capacità k e tipi di oggetto disponibili $\{1, 2, \ldots, j\}$ si costruisce combinando il numero d scelto di copie del tipo di oggetto j con una soluzione ottima del corrispondente sottoproblema residuo

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

Sottoproblemi sovrapposti

- Scelte diverse del numero di copie di un tipo di oggetto, fatte eventualmente per valori diversi di j e k, possono portare allo **stesso sottoproblema**
- Quindi lo stesso valore $v(j, k)$ può dover essere calcolato più volte durante la valutazione ricorsiva della ricorrenza

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

Primo metodo: l'approccio top-down

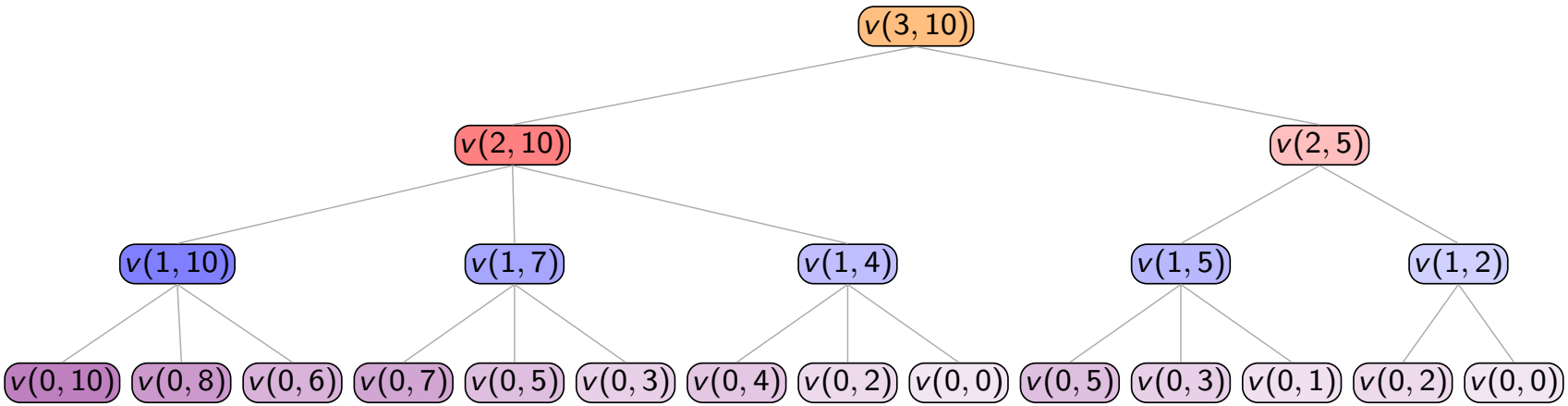
- Un **primo metodo** per calcolare il valore ottimo $v(n, c)$ dell'istanza del BKP è l'**approccio top-down**
- Il metodo parte dalla chiamata con $j = n$ e $k = c$ e valuta ricorsivamente la ricorrenza
- Ogni chiamata considera il tipo di oggetto j e ne prova ogni numero ammissibile d di copie, finché non resta alcun tipo di oggetto o la capacità residua è uguale a 0

L'albero di ricorsione

- L'**albero di ricorsione** della funzione v 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 effettuate da quella chiamata
- Le foglie rappresentano i casi base

Esempio: albero di ricorsione

Consideriamo l'istanza del BKP degli esempi. Questo è l'albero di ricorsione dell'approccio top-down per calcolare $v(3, 10)$:



La funzione v viene chiamata 22 volte, e gli stessi sottoproblemi vengono risolti più volte: $v(0, 5)$, $v(0, 3)$, $v(0, 2)$ e $v(0, 0)$ vengono chiamate 2 volte ciascuna. Tutte le altre chiamate vengono effettuate una sola volta.

Ordine di calcolo

- Il valore $v(j, k)$ si può calcolare una volta noti i valori $v(j-1, k')$ per ogni capacità residua $k' \leq k$
- Quindi, fissato j, tutti i valori $v(j, k)$ si possono calcolare considerando i valori di capacità k in ordine crescente

Vediamo questo ordine sull'istanza degli esempi.

Esempio: calcolo dei valori della funzione v (1/4)

Consideriamo l'istanza del BKP degli esempi, con uno zaino di capacità $c = 10$ e i seguenti profitti, pesi e limiti:

	$j = 1$	$j = 2$	$j = 3$
p_j	6	10	12
w_j	2	3	5
b_j	2	2	1

I valori della funzione v si calcolano come segue:

- I casi base danno $v(0, k) = 0$ per ogni $k \geq 0$.

Esempio: calcolo dei valori della funzione v (2/4)

- Calcolati i casi base, possiamo calcolare $v(1,4)$, provando $d = 0, 1, 2$ copie del tipo di oggetto 1 (perché $b_1 = 2$ e $\lfloor 4/2 \rfloor = 2$):

$$\begin{aligned} v(1,4) &= \max\{v(0,4),\ p_1 + v(0,2),\ 2p_1 + v(0,0)\} \\ &= \max\{0,\ 6,\ 12\} = 12 \end{aligned}$$

- Possiamo calcolare anche $v(1,10)$, provando $d = 0, 1, 2$ copie (perché $b_1 = 2$ limita il numero di copie anche se $\lfloor 10/2 \rfloor = 5$):

$$\begin{aligned} v(1,10) &= \max\{v(0,10),\ p_1 + v(0,8),\ 2p_1 + v(0,6)\} \\ &= \max\{0,\ 6,\ 12\} = 12 \end{aligned}$$

Esempio: calcolo dei valori della funzione v (3/4)

- Allo stesso modo calcoliamo $v(1,7)$, $v(1,5)$ e $v(1,2)$:

$$\begin{aligned} v(1,7) &= \max\{v(0,7),\ p_1 + v(0,5),\ 2p_1 + v(0,3)\} \\ &= \max\{0,\ 6,\ 12\} = 12 \end{aligned}$$

$$\begin{aligned} v(1,5) &= \max\{v(0,5),\ p_1 + v(0,3),\ 2p_1 + v(0,1)\} \\ &= \max\{0,\ 6,\ 12\} = 12 \end{aligned}$$

$$v(1,2) = \max\{v(0,2),\ p_1 + v(0,0)\} = \max\{0,\ 6\} = 6$$

- Calcolati $v(1,5)$ e $v(1,2)$, possiamo calcolare $v(2,5)$, provando $d = 0, 1$ copie del tipo di oggetto 2 (perché $b_2 = 2$ ma $\lfloor 5/3 \rfloor = 1$):

$$v(2,5) = \max\{v(1,5),\ p_2 + v(1,2)\} = \max\{12,\ 10 + 6\} = 16$$

Esempio: calcolo dei valori della funzione v (4/4)

- Calcolati $v(1,10)$, $v(1,7)$ e $v(1,4)$, possiamo calcolare $v(2,10)$, provando $d = 0, 1, 2$ copie del tipo di oggetto 2:

$$\begin{aligned} v(2,10) &= \max\{v(1,10),\ p_2 + v(1,7),\ 2p_2 + v(1,4)\} \\ &= \max\{12,\ 10+12,\ 20+12\} = 32 \end{aligned}$$

- Infine, calcolati $v(2,10)$ e $v(2,5)$, possiamo calcolare $v(3,10)$, provando $d = 0, 1$ copie del tipo di oggetto 3 (perché $b_3 = 1$):

$$\begin{aligned} v(3,10) &= \max\{v(2,10),\ p_3 + v(2,5)\} \\ &= \max\{32,\ 12+16\} = 32 \end{aligned}$$

La tabella di lookup

- Possiamo evitare di risolvere ripetutamente gli stessi sottoproblemi memorizzando il valore di ciascun sottoproblema appena viene calcolato
- I valori sono memorizzati in una **tabella di lookup**

Risolvere i sottoproblemi in ordine

- La soluzione di ogni sottoproblema dipende solo da sottoproblemi con un valore di j minore
- Possiamo risolvere i sottoproblemi in ordine crescente di j e, per ogni valore fissato di j, in ordine crescente di capacità, partendo dai casi base $v(0, k) = 0$ e $v(j, 0) = 0$ fino ad arrivare a $v(n, c)$
- Questa osservazione porta in modo naturale a un **metodo molto più efficiente** per calcolare il valore ottimo $v(n, c)$ dell'istanza del BKP

Secondo metodo: l'approccio bottom-up

- Un **secondo metodo** per calcolare il valore ottimo $v(n, c)$ dell'istanza del BKP è l'**approccio bottom-up**
- Il metodo parte dai casi base e valuta la ricorrenza per valori crescenti di $j \in \{1, 2, \ldots, n\}$ e $k \in \{0, 1, \ldots, c\}$
- Ogni valore $v(j, k)$ si calcola usando valori già calcolati, fino a ottenere il valore $v(n, c)$ dell'istanza del BKP

Segue lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**.

Algorithm: BKP_DP (1/2)

Input : Un'istanza del BKP data da n, dai vettori $p[1..n]$, $w[1..n]$, $b[1..n]$ e dalla capacità c

Output: Il valore ottimo dell'istanza del BKP

```
1 function BKP_DP(n, p, w, b, c)
2     v := new array[0..n][0..c]
3     for k := 0 to c do
4         v[0][k] ← 0
5     end
```

Algorithm: BKP_DP (2/2)

```
6     for j := 1 to n do
7         for k := 0 to c do
8             m := 0
9             for d := 0 to min{b[j], ⌊k/w[j]⌋} do
10                if d p[j] + v[j − 1][k − d w[j]] > m then
11                    m ← d p[j] + v[j − 1][k − d w[j]]
12                end
13            end
14            v[j][k] ← m
15        end
16    end
17    return v[n][c]
18 end
```

Come funziona BKP_DP (1/2)

- La matrice $v[0..n][0..c]$ è la **tabella di lookup**: memorizza i valori ottimi di tutti i sottoproblemi
- Il primo ciclo inizializza i valori $v[0][k] = 0$ per tutte le capacità $k \in \{0, 1, \ldots, c\}$, corrispondenti al caso in cui nessun tipo di oggetto è disponibile
- Il ciclo esterno, su $j \in \{1, 2, \ldots, n\}$, calcola le righe della matrice in ordine crescente di tipo di oggetto
- Il ciclo interno, su $k \in \{0, 1, \ldots, c\}$, calcola il valore ottimo $v[j][k]$ per ogni capacità k

Come funziona BKP_DP (2/2)

- Il ciclo più interno, su $d \in \{0, 1, \ldots, \min\{b[j], \lfloor k/w[j] \rfloor\}\}$, prova ogni numero ammissibile di copie del tipo di oggetto j e conserva in m il valore massimo trovato finora
- Al termine del ciclo più interno, $v[j][k] \leftarrow m$ memorizza il valore ottimo $v[j][k]$
- Infine, il valore ottimo dell'istanza del BKP viene restituito come $v[n][c]$

Esempio: esecuzione dell'algoritmo di PD (1/8)

Consideriamo l'istanza del BKP degli esempi, con uno zaino di capacità $c = 10$ e i seguenti profitti, pesi e limiti:

	$j = 1$	$j = 2$	$j = 3$
p_j	6	10	12
w_j	2	3	5
b_j	2	2	1

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

Esempio: esecuzione dell'algoritmo di PD (2/8)

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

Esempio: esecuzione dell'algoritmo di PD (3/8)

j	k	$v[j][k]$	Azione
1	0	0	$\max\{v[0][0]\} = 0$
1	1	0	$\max\{v[0][1]\} = 0$
1	2	6	$\max\{v[0][2],\ p[1]+v[0][0]\} = \max\{0, 6+0\} = 6$
1	3	6	$\max\{v[0][3],\ p[1]+v[0][1]\} = \max\{0, 6+0\} = 6$
1	4	12	$\max\{v[0][4],\ p[1]+v[0][2],\ 2p[1]+v[0][0]\}$ $= \max\{0, 6, 12\} = 12$
1	5	12	$\max\{v[0][5],\ p[1]+v[0][3],\ 2p[1]+v[0][1]\}$ $= \max\{0, 6, 12\} = 12$

Esempio: esecuzione dell'algoritmo di PD (4/8)

j	k	$v[j][k]$	Azione
1	6	12	$\max\{v[0][6],\ p[1]+v[0][4],\ 2p[1]+v[0][2]\}$ $= \max\{0, 6, 12\} = 12$
1	7	12	$\max\{v[0][7],\ p[1]+v[0][5],\ 2p[1]+v[0][3]\}$ $= \max\{0, 6, 12\} = 12$
1	8	12	$\max\{v[0][8],\ p[1]+v[0][6],\ 2p[1]+v[0][4]\}$ $= \max\{0, 6, 12\} = 12$
1	9	12	$\max\{v[0][9],\ p[1]+v[0][7],\ 2p[1]+v[0][5]\}$ $= \max\{0, 6, 12\} = 12$
1	10	12	$\max\{v[0][10],\ p[1]+v[0][8],\ 2p[1]+v[0][6]\}$ $= \max\{0, 6, 12\} = 12$

Esempio: esecuzione dell'algoritmo di PD (5/8)

j	k	$v[j][k]$	Azione
2	0	0	$\max\{v[1][0]\} = 0$
2	1	0	$\max\{v[1][1]\} = 0$
2	2	6	$\max\{v[1][2]\} = 6$
2	3	10	$\max\{v[1][3],\ p[2]+v[1][0]\} = \max\{6, 10+0\} = 10$
2	4	12	$\max\{v[1][4],\ p[2]+v[1][1]\} = \max\{12, 10+0\} = 12$
2	5	16	$\max\{v[1][5],\ p[2]+v[1][2]\} = \max\{12, 10+6\} = 16$

Esempio: esecuzione dell'algoritmo di PD (6/8)

j	k	$v[j][k]$	Azione
2	6	20	$\max\{v[1][6],\ p[2]+v[1][3],\ 2p[2]+v[1][0]\}$ $= \max\{12, 16, 20\} = 20$
2	7	22	$\max\{v[1][7],\ p[2]+v[1][4],\ 2p[2]+v[1][1]\}$ $= \max\{12, 22, 20\} = 22$
2	8	26	$\max\{v[1][8],\ p[2]+v[1][5],\ 2p[2]+v[1][2]\}$ $= \max\{12, 22, 26\} = 26$
2	9	26	$\max\{v[1][9],\ p[2]+v[1][6],\ 2p[2]+v[1][3]\}$ $= \max\{12, 22, 26\} = 26$
2	10	32	$\max\{v[1][10],\ p[2]+v[1][7],\ 2p[2]+v[1][4]\}$ $= \max\{12, 22, 32\} = 32$

Esempio: esecuzione dell'algoritmo di PD (7/8)

j	k	$v[j][k]$	Azione
3	0	0	$\max\{v[2][0]\} = 0$
3	1	0	$\max\{v[2][1]\} = 0$
3	2	6	$\max\{v[2][2]\} = 6$
3	3	10	$\max\{v[2][3]\} = 10$
3	4	12	$\max\{v[2][4]\} = 12$
3	5	16	$\max\{v[2][5],\ p[3]+v[2][0]\} = \max\{16, 12+0\} = 16$
3	6	20	$\max\{v[2][6],\ p[3]+v[2][1]\} = \max\{20, 12+0\} = 20$
3	7	22	$\max\{v[2][7],\ p[3]+v[2][2]\} = \max\{22, 12+6\} = 22$
3	8	26	$\max\{v[2][8],\ p[3]+v[2][3]\} = \max\{26, 12+10\} = 26$
3	9	26	$\max\{v[2][9],\ p[3]+v[2][4]\} = \max\{26, 12+12\} = 26$
3	10	32	$\max\{v[2][10],\ p[3]+v[2][5]\} = \max\{32, 12+16\} = 32$

Esempio: esecuzione dell'algoritmo di PD (8/8)

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

$v[j][k]$	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$j=0$	0	0	0	0	0	0
$j=1$	0	0	6	6	12	12
$j=2$	0	0	6	10	12	16
$j=3$	0	0	6	10	12	16

$v[j][k]$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$j=0$	0	0	0	0	0
$j=1$	12	12	12	12	12
$j=2$	20	22	26	26	32
$j=3$	20	22	26	26	32

Passo 2: calcolo di una soluzione ottima

- Dopo l'esecuzione dell'algoritmo di PD, la matrice v contiene il valore ottimo $v[j][k]$ per ogni tipo di oggetto $j \in \{0, 1, \ldots, n\}$ e ogni valore di capacità $k \in \{0, 1, \ldots, c\}$
- Per calcolare una soluzione ottima, cioè il vettore intero $(x_1, x_2, \ldots, x_n)$, è necessario memorizzare alcune **informazioni aggiuntive**

La matrice s dei numeri di copie

- Per ogni tipo di oggetto j e valore di capacità k, oltre al valore ottimo $v[j][k]$, memorizziamo $s[j][k]$, il numero di copie del tipo di oggetto j selezionate in una soluzione ottima del sottoproblema ristretto ai tipi di oggetto $\{1, 2, \ldots, j\}$ e alla capacità k

- Più precisamente, $s[j][k]$ è uguale al valore d che massimizza il membro destro della ricorrenza nel calcolo di $v[j][k]$

Segue lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**, con la memorizzazione delle informazioni necessarie a costruire una soluzione ottima.

Algorithm: BKP_DP_InfoSol (1/2)

Input : Un'istanza del BKP data da n, dai vettori $p[1..n]$, $w[1..n]$, $b[1..n]$ e dalla capacità c

Output : Il valore ottimo dell'istanza del BKP e la matrice dei numeri di copie $s[0..n][0..c]$

```
1 function BKP_DP_InfoSol(n, p, w, b, c)
2     v := new array[0..n][0..c]; s := new array[0..n][0..c]
3     for k := 0 to c do
4     |     v[0][k] ← 0; s[0][k] ← 0
5     end
```

Algorithm: BKP_DP_InfoSol (2/2)

```
6     for j := 1 to n do
7         for k := 0 to c do
8             m := 0; t := 0
9             for d := 0 to min{b[j], ⌊k/w[j]⌋} do
10                if d p[j] + v[j − 1][k − d w[j]] > m then
11                    m ← d p[j] + v[j − 1][k − d w[j]]; t ← d
12                end
13            end
14            v[j][k] ← m; s[j][k] ← t
15        end
16    end
17    return v[n][c], s
18 end
```

Memorizzare le decisioni

- L'algoritmo di PD usa la variabile temporanea aggiuntiva t per tenere traccia della decisione presa quando m viene aggiornato
- Se vengono selezionate d copie del tipo di oggetto j, $t \leftarrow d$ memorizza questa decisione
- Al termine del ciclo più interno, $s[j][k] \leftarrow t$ memorizza il numero di copie del tipo di oggetto j usate, insieme al valore ottimo $v[j][k] \leftarrow m$

Esempio: calcolo delle informazioni aggiuntive (1/7)

Consideriamo l'istanza del BKP degli esempi, con uno zaino di capacità $c = 10$ e i seguenti profitti, pesi e limiti:

	$j = 1$	$j = 2$	$j = 3$
p_j	6	10	12
w_j	2	3	5
b_j	2	2	1

- Per ogni cella con $j \geq 1$ le tabelle delle slide seguenti riportano i valori $d\, p[j] + v[j-1][k - d\, w[j]]$ per $d = 0, 1, \ldots, \min\{b[j], \lfloor k/w[j] \rfloor\}$, in quest'ordine, e il numero t di copie memorizzato in $s[j][k]$

Esempio: calcolo delle informazioni aggiuntive (2/7)

Due osservazioni sulle tabelle:

- In caso di pareggio si conserva il d minimo, perché m viene aggiornato solo quando il valore è strettamente maggiore
- La riga $j = 0$ è $s[0][k] = 0$ per ogni k

Esempio: calcolo delle informazioni aggiuntive (3/7)

j	k	$s[j][k]$	Valori per $d = 0, 1, \ldots$
1	0	0	0
1	1	0	0
1	2	1	0, 6
1	3	1	0, 6
1	4	2	0, 6, 12
1	5	2	0, 6, 12
1	6	2	0, 6, 12
1	7	2	0, 6, 12
1	8	2	0, 6, 12
1	9	2	0, 6, 12
1	10	2	0, 6, 12

Esempio: calcolo delle informazioni aggiuntive (4/7)

j	k	$s[j][k]$	Valori per $d = 0, 1, \ldots$
2	0	0	0
2	1	0	0
2	2	0	6
2	3	1	6, 10
2	4	0	12, 10
2	5	1	12, 16
2	6	2	12, 16, 20
2	7	1	12, 22, 20
2	8	2	12, 22, 26
2	9	2	12, 22, 26
2	10	2	12, 22, 32

Esempio: calcolo delle informazioni aggiuntive (5/7)

j	k	$s[j][k]$	Valori per $d = 0, 1, \ldots$
3	0	0	0
3	1	0	0
3	2	0	6
3	3	0	10
3	4	0	12
3	5	0	16, 12
3	6	0	20, 12
3	7	0	22, 18
3	8	0	26, 22
3	9	0	26, 24
3	10	0	32, 28

Esempio: calcolo delle informazioni aggiuntive (6/7)

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

$s[j][k]$	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$j=0$	0	0	0	0	0	0
$j=1$	0	0	1	1	2	2
$j=2$	0	0	0	1	0	1
$j=3$	0	0	0	0	0	0

$s[j][k]$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$j=0$	0	0	0	0	0
$j=1$	2	2	2	2	2
$j=2$	2	1	2	2	2
$j=3$	0	0	0	0	0

Esempio: calcolo delle informazioni aggiuntive (7/7)

Per esempio:

- $s[1][4] = 2$, perché la ricorrenza seleziona $d = 2$ copie del tipo di oggetto 1 nel calcolo di $v[1][4] = 12$
- $s[2][10] = 2$, perché vengono selezionate $d = 2$ copie del tipo di oggetto 2 nel calcolo di $v[2][10] = 32$
- Poiché $s[3][10] = 0$, nella soluzione ottima dell'istanza originale non si usa alcuna copia del tipo di oggetto 3

Ricostruire una soluzione ottima

- Calcolata la matrice s, si può ricostruire una soluzione ottima scorrendo i tipi di oggetto da $j = n$ fino a $j = 1$, partendo dalla capacità $k = c$ e da $x_j = 0$ per ogni $j \in \{1, 2, \ldots, n\}$:

$$x_j \leftarrow s[j][k], \qquad k \leftarrow k - s[j][k]\, w_j.$$

- Si ottiene così il vettore $(x_1, x_2, \ldots, x_n)$, in cui ogni componente x_j è il numero di copie del tipo di oggetto j in una soluzione ottima

Algorithm: BKP_Reconstruct

Input : Il numero n di tipi di oggetto, la capacità c dello zaino, il vettore $w[1..n]$ dei pesi e la matrice $s[0..n][0..c]$

Output: Una soluzione ottima dell'istanza del BKP

```
1 function BKP_Reconstruct(n, c, w, s)
2     x := new array[1..n]
3     k := c
4     for j := n downto 1 do
5         x[j] ← s[j][k]
6         k ← k − x[j] w[j]
7     end
8     return x
9 end
```

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

Dopo l'esecuzione dell'algoritmo di PD sull'istanza degli esempi, abbiamo la seguente matrice s:

$s[j][k]$	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$j=0$	0	0	0	0	0	0
$j=1$	0	0	1	1	2	2
$j=2$	0	0	0	1	0	1
$j=3$	0	0	0	0	0	0

$s[j][k]$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$j=0$	0	0	0	0	0
$j=1$	2	2	2	2	2
$j=2$	2	1	2	2	2
$j=3$	0	0	0	0	0

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

Possiamo ricostruire una soluzione ottima partendo da $j = 3$ e $k = 10$ come segue:

Iterazione	j	k	$s[j][k]$	$x[1]$	$x[2]$	$x[3]$	Azione
–	–	–	–	0	0	0	$k = 10$
1	3	10	0	0	0	0	$x[3] \leftarrow 0,\ k = 10$
2	2	10	2	0	2	0	$x[2] \leftarrow 2,\ k \leftarrow 10 - 2 \cdot 3 = 4$
3	1	4	2	2	2	0	$x[1] \leftarrow 2,\ k \leftarrow 4 - 2 \cdot 2 = 0$

Quindi il vettore

$$(2, 2, 0)$$

è una soluzione ottima. Il peso totale corrispondente è $2 \cdot 2 + 2 \cdot 3 = 4 + 6 = 10 = c$. Il valore ottimo corrispondente è $v(3, 10) = 32$.

Numero di passi

- Sia $b_{\max} = \max\{b[j] : j \in \{1, \ldots, n\}\}$, e indichiamo con $S(n, c, b_{\max})$ il numero di passi eseguiti da `BKP_DP` nel caso peggiore, dove n è il numero di tipi di oggetto e c è la capacità dello zaino

- Calcoliamo $S(n, c, b_{\max})$ contando quante volte vengono eseguite le righe dello pseudocodice di `BKP_DP`

Il ciclo più interno

- Per ogni coppia (j, k) il ciclo più interno su d esegue $\min\{b_j, \lfloor k/w_j \rfloor\} + 1$ iterazioni, cioè al più $b_{\max} + 1$
- Contiamo $b_{\max} + 1$ iterazioni per ogni coppia (j, k)
- Supponiamo che la condizione alla riga 10 sia vera a ogni iterazione
- Così $S(n, c, b_{\max})$ è un **upper bound** sul numero di passi

Nelle due slide seguenti richiamiamo lo pseudocodice di `BKP_DP`, con i numeri di riga.

Algorithm: BKP_DP (1/2)

Input : Un'istanza del BKP data da n, dai vettori $p[1..n]$, $w[1..n]$, $b[1..n]$ e dalla capacità c

Output: Il valore ottimo dell'istanza del BKP

```
1 function BKP_DP(n, p, w, b, c)
2     v := new array[0..n][0..c]
3     for k := 0 to c do
4     |     v[0][k] ← 0
5     end
```

Algorithm: BKP_DP (2/2)

```
6     for j := 1 to n do
7         for k := 0 to c do
8             m := 0
9             for d := 0 to min{b[j], ⌊k/w[j]⌋} do
10                if d p[j] + v[j − 1][k − d w[j]] > m then
11                    m ← d p[j] + v[j − 1][k − d w[j]]
12                end
13            end
14            v[j][k] ← m
15        end
16    end
17    return v[n][c]
18 end
```

Contare le esecuzioni (1/2)

Il numero di esecuzioni delle righe rilevanti è, al più:

- La riga 2 viene eseguita 1 volta.
- La riga 3 viene eseguita $c + 1$ volte.
- La riga 4 viene eseguita $c + 1$ volte.
- La riga 6 viene eseguita n volte.
- La riga 7 viene eseguita $n(c + 1)$ volte.
- La riga 8 viene eseguita $n(c + 1)$ volte.

Contare le esecuzioni (2/2)

Il numero di esecuzioni delle righe rilevanti è, al più:

- La riga 9 viene eseguita al più $n(c+1)(b_{\max}+1)$ volte.
- La riga 10 viene eseguita al più $n(c+1)(b_{\max}+1)$ volte.
- La riga 11 viene eseguita al più $n(c+1)(b_{\max}+1)$ volte.
- La riga 14 viene eseguita $n(c+1)$ volte.
- La riga 17 viene eseguita 1 volta.

Il numero totale di passi

Quindi il numero totale di passi nel caso peggiore è

$$1 + (c+1) + (c+1) + n + n(c+1) + n(c+1) \\ + 3n(c+1)(b_{\max}+1) + n(c+1) + 1.$$

Pertanto,

$$S(n, c, b_{\max}) = 3n(c+1)(b_{\max}+1) + 3n(c+1) + n + 2c + 4.$$

Di conseguenza,

$$S(n, c, b_{\max}) = \Theta(nc\, b_{\max}).$$

Verifica dell'ordine di crescita

Infatti,

$$\lim_{n,c,b_{\max}\to\infty} \frac{3n(c+1)(b_{\max}+1)+3n(c+1)+n+2c+4}{nc\,b_{\max}}$$

$$= \lim_{n,c,b_{\max}\to\infty} \left(\frac{3(c+1)(b_{\max}+1)}{c\,b_{\max}} + \frac{3(c+1)}{c\,b_{\max}} + \frac{1}{c\,b_{\max}} + \frac{2}{n\,b_{\max}} + \frac{4}{nc\,b_{\max}} \right)$$

$$= 3 \in \mathbb{R}_{>0}.$$

Numero di passi dell'algoritmo di PD

L'algoritmo di PD esegue

$$S(n, c, b_{\max}) = \Theta(nc\, b_{\max})$$

passi nel caso peggiore.

- La versione `BKP_DP_InfoSol` memorizza la matrice aggiuntiva s ed esegue solo un numero costante di operazioni in più negli stessi cicli
- Quindi il suo numero di passi resta $\Theta(nc\, b_{\max})$

Quando il bound è stretto?

- Sostituire ogni limite b_j con $\min\{b_j, \lfloor c/w_j \rfloor\}$ non cambia il problema, quindi possiamo supporre $b_{\max} \leq c$
- Sotto questa ipotesi il bound $\Theta(nc\, b_{\max})$ è **stretto**: è raggiunto dalle istanze in cui tutti i pesi sono uguali a 1 e tutti i limiti sono uguali a $b_{\max}$
- Infatti il ciclo più interno esegue allora

$$\sum_{k=0}^{c} (\min\{b_{\max}, k\} + 1) \geq (c+1)\, b_{\max}/2$$

iterazioni per ogni tipo di oggetto

Riepilogo (1/2)

Passo 1: il valore ottimo:

- $v(j, k)$ prova ogni numero $d \in \{0, 1, \ldots, \min\{b_j, \lfloor k/w_j \rfloor\}\}$ di copie del tipo di oggetto j; casi base $v(0, k) = v(j, 0) = 0$
- Valgono la sottostruttura ottima e i sottoproblemi sovrapposti: l'approccio top-down chiama v 22 volte sull'esempio, l'approccio bottom-up riempie la tabella di lookup riga per riga
- `BKP_DP` restituisce $v[n][c]$: sull'esempio, $v(3, 10) = 32$

Passo 2: una soluzione ottima:

- $s[j][k]$ memorizza il numero di copie scelto (il d minimo in caso di pareggio)
- All'indietro da (n, c): $x_j \leftarrow s[j][k]$, $k \leftarrow k - s[j][k]\, w_j$; sull'esempio, $(2, 2, 0)$

Riepilogo (2/2)

Numero di passi:

- Nel caso peggiore $S(n, c, b_{\max}) = 3n(c+1)(b_{\max}+1) + 3n(c+1) + n + 2c + 4 = \Theta(nc\, b_{\max})$
- `BKP_DP_InfoSol` ha lo stesso ordine di crescita
- Con $b_{\max} \leq c$ il bound è stretto: tutti i pesi uguali a 1 e tutti i limiti uguali a $b_{\max}$

Errori tipici

Introduzione: errori tipici

- Elenchiamo gli errori che si commettono più spesso con l'algoritmo di PD per il BKP
- Riguardano l'intervallo del numero di copie, la ricostruzione, il conteggio dei passi e i pareggi
- Conoscerli aiuta a eseguire correttamente l'algoritmo a mano negli esercizi

Osservazione 1: Errori tipici (1/2)

- ***Provare più di b_j copie.*** *Il numero di copie del tipo di oggetto j varia in $\{0, 1, \ldots, \min\{b_j, \lfloor k/w_j \rfloor\}\}$: contano sia il limite sia la capacità.*
- ***Ricostruire con l'aggiornamento sbagliato.*** *Dopo aver letto $s[j][k]$ la capacità diminuisce di $s[j][k]$ w_j, non di w_j, e l'indice del tipo di oggetto diminuisce sempre di uno.*

Osservazione: Errori tipici (2/2)

- ***Contare i passi con $b_{\max}$ per ogni cella.*** *Il ciclo più interno ha $\min\{b_j, \lfloor k/w_j \rfloor\} + 1$ iterazioni, quindi $b_{\max} + 1$ è solo un upper bound.*
- ***Risolvere i pareggi in modo incoerente.*** *L'algoritmo di PD aggiorna m solo con un valore strettamente maggiore, quindi $s[j][k]$ conserva il numero minimo di copie che raggiunge il massimo.*

Riepilogo

Errori tipici:

- Il numero di copie è al più $\min\{b_j, \lfloor k/w_j \rfloor\}$: contano sia il limite sia la capacità
- Nella ricostruzione la capacità diminuisce di $s[j][k]\, w_j$, e j diminuisce sempre di uno
- $b_{\max} + 1$ iterazioni per cella è solo un upper bound
- Pareggi: m si aggiorna solo con un valore strettamente maggiore, quindi $s[j][k]$ conserva il d minimo

In sintesi

Introduzione: in sintesi

- Raccogliamo le idee chiave del capitolo
- Chiudiamo con il quadro riassuntivo dell'istanza degli esempi: i bound trovati in questo capitolo

Riepilogo (1/2)

- Il BKP generalizza sia il KP ($b_j = 1$) sia l'UKP ($b_j = \lfloor c/w_j \rfloor$); la ricorrenza prova ogni numero d di copie del tipo di oggetto j.
- La matrice $v(j, k)$ si riempie in $\Theta(nc\, b_{\max})$ passi, un numero pseudo-polinomiale.
- La matrice s memorizza il numero di copie di ciascun tipo di oggetto, e una soluzione ottima si legge all'indietro a partire da (n, c).
- Sostituire b_j con $\min\{b_j, \lfloor c/w_j \rfloor\}$ lascia invariato il problema e dà $b_{\max} \leq c$.

Riepilogo (2/2)

Quadro riassuntivo dell'esempio del BKP. L'istanza degli esempi ($c = 10$, tre tipi di oggetto):

Metodo	Lower bound	Upper bound	Origine
Algoritmo di PD	32	32	valore ottimo, soluzione $(2, 2, 0)$