PD per il problema del taglio delle aste

Indice

Istanze e soluzioni ammissibili
    Codifica delle soluzioni ammissibili

Un primo algoritmo di PD
    Numero di passi del primo algoritmo di PD

Un secondo algoritmo di PD
    Numero di passi del secondo algoritmo di PD

Errori tipici

In sintesi

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- enunciare l'RCP, codificare una soluzione come vettore di numeri di pezzi e spiegare perché schemi di taglio e vettori sono oggetti diversi;
- scrivere la ricorrenza $v(\ell)$ e spiegarla, caso base compreso;
- eseguire a mano l'algoritmo di PD con l'approccio top-down e con quello bottom-up, e con il miglior valore corrente m;
- ricostruire una soluzione ottima dal vettore s, per l'asta e per ogni asta più corta;
- scrivere la seconda ricorrenza $v(j, \ell)$ sulle coppie (lunghezza massima dei pezzi, lunghezza della asta) e contare i passi dell'algoritmo di PD.

Prerequisiti

- Da questo modulo, il capitolo «Principi della programmazione dinamica»: sottostruttura ottima, sottoproblemi sovrapposti, relazioni di ricorrenza, casi base e approcci top-down e bottom-up.
- Da questo modulo, il capitolo «PD per il problema dello zaino 0/1»: le coppie (oggetto, capacità) e la ricostruzione di una soluzione, che ritornano qui.
- Dal modulo *Algoritmi*: come leggere lo pseudocodice e come contare i passi di un algoritmo («Tempo di esecuzione»).

Il problema di questo capitolo

Il problema di questo capitolo è il
problema del taglio delle aste (RCP):

la sua definizione è nella sua pagina.

- Descriviamo le istanze e le soluzioni ammissibili dell'RCP
- Scriviamo due algoritmi di PD: il primo sulle lunghezze dell'asta ℓ, il secondo sulle coppie (j, ℓ) (lunghezza massima dei pezzi, lunghezza dell'asta)

Istanze e soluzioni ammissibili

Introduzione: Istanze e soluzioni ammissibili

- Presentiamo l'istanza dell'RCP che usiamo in tutto il capitolo
- Contiamo le soluzioni ammissibili di un'istanza dell'RCP, e vediamo perché enumerarle diventa presto impraticabile
- Codifichiamo una soluzione ammissibile come vettore di numeri di pezzi, e vediamo perché schemi di taglio diversi possono avere lo stesso vettore
- È il punto di partenza dei due algoritmi di PD delle sezioni successive

Esempio: istanza dell'RCP

Consideriamo un'asta di lunghezza $n = 4$ e un insieme $\{1, 2, 3, 4\}$ di 4 possibili lunghezze dei pezzi con i seguenti profitti:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
p_ℓ	1	5	8	9

Chiamiamo questa istanza dell'RCP l'istanza di riferimento: la usiamo in tutti gli esempi del capitolo.

Quante soluzioni ammissibili?

Il numero di possibili soluzioni dell'RCP è 2^{n-1},

perché possiamo tagliare un'asta di lunghezza n
in 2^{n-1} modi diversi,

con una scelta indipendente di tagliare (o no)
in ogni possibile posizione di taglio.

- Il numero di possibili soluzioni dell'RCP cresce esponenzialmente con la lunghezza n dell'asta:

$$2^{n-1} = O(2^n).$$

- Quindi enumerare tutte le possibili soluzioni diventa presto impraticabile al crescere di n.

Soluzioni ammissibili

Una *soluzione ammissibile* dell'RCP è un modo di tagliare l'asta in pezzi, tale che la lunghezza totale dei pezzi sia uguale alla lunghezza dell'asta.

- Una soluzione ammissibile dell'RCP si può descrivere con un vettore soluzione intero

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

dove ogni componente ha il seguente significato:

$$x_\ell = \text{numero di pezzi di lunghezza } \ell, \qquad \forall \ell \in \{1, 2, \ldots, n\}.$$

Codifica delle soluzioni ammissibili

- Il numero di pezzi è

$$\sum_{\ell=1}^{n} x_\ell .$$

- Poiché la soluzione è ammissibile, la lunghezza totale dei pezzi deve essere uguale alla lunghezza dell'asta:

$$\sum_{\ell=1}^{n} \ell \, x_\ell = n.$$

- Il profitto totale di una soluzione ammissibile è

$$\sum_{\ell=1}^{n} p_\ell x_\ell .$$

Schemi di taglio e vettori

- Due o più schemi di taglio possono avere la stessa rappresentazione vettoriale: il vettore registra quanti pezzi di ogni lunghezza si ottengono, ma non specifica l'ordine in cui i pezzi vengono tagliati.

- Schemi di taglio con la stessa rappresentazione vettoriale hanno lo stesso profitto totale, perché l'ordine in cui i pezzi vengono tagliati non cambia i pezzi ottenuti e quindi non cambia il profitto totale.

Ora: tutte le soluzioni ammissibili dell'istanza di riferimento.

Esempio: enumerazione di tutte le soluzioni dell'RCP (1/5)

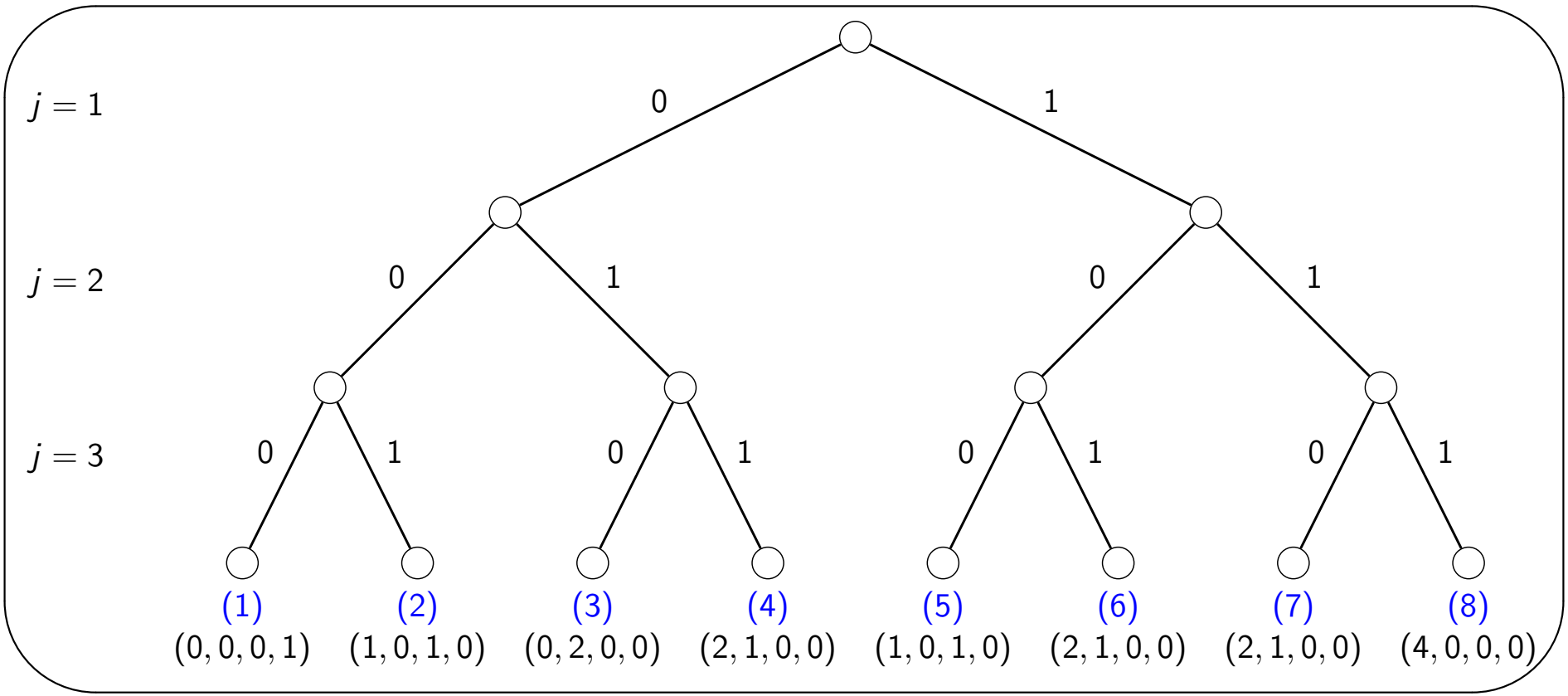
Consideriamo l'istanza di riferimento, con un'asta di lunghezza $n = 4$ e i seguenti profitti delle possibili lunghezze dei pezzi:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
p_ℓ	1	5	8	9

- Ci sono $2^{4-1} = 8$ soluzioni ammissibili, corrispondenti ai diversi modi di tagliare l'asta.
- Nell'albero binario della slide successiva ogni livello corrisponde a una possibile posizione di taglio: un arco con etichetta 1 indica che l'asta viene tagliata a quella lunghezza, mentre un arco con etichetta 0 indica che non viene tagliata a quella lunghezza.
- Ogni foglia rappresenta uno schema di taglio completo e riporta il vettore corrispondente.

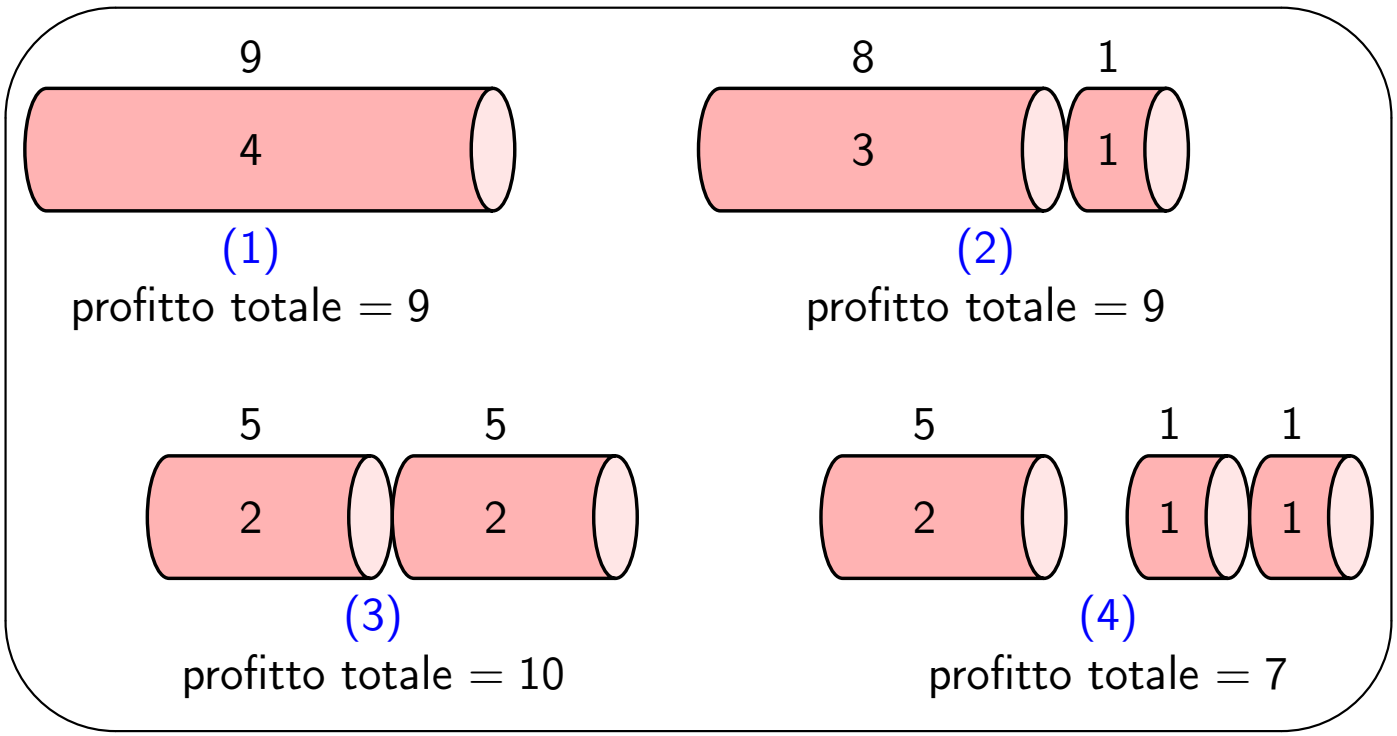
Esempio: enumerazione di tutte le soluzioni dell'RCP (2/5)

L'albero binario delle decisioni di taglio nelle posizioni $j = 1, 2, 3$:



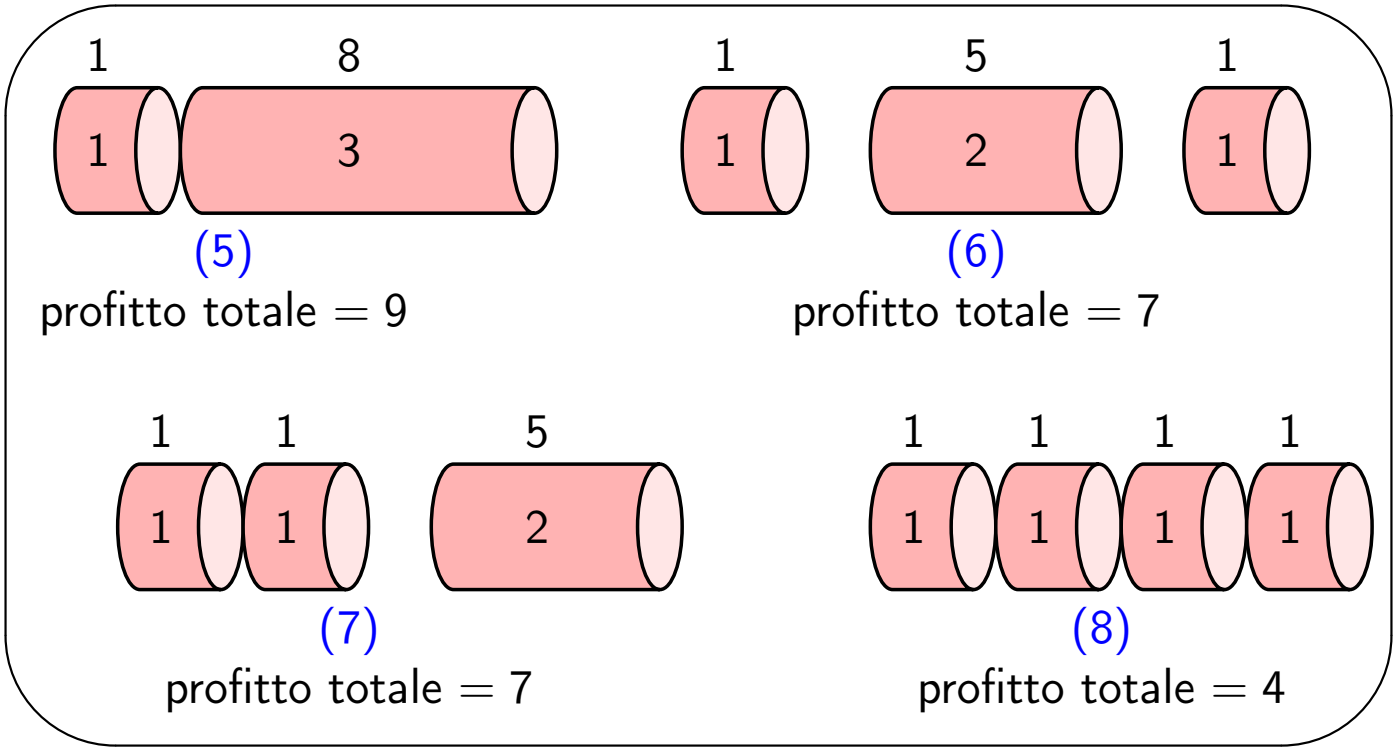
Esempio: enumerazione di tutte le soluzioni dell'RCP (3/5)

Le figure seguenti mostrano tutti i possibili schemi di taglio associati alle foglie dell'albero, tramite le etichette $(1), (2), \ldots, (8)$. Per ogni pezzo ottenuto, il numero scritto dentro il pezzo è la sua lunghezza, mentre il numero sopra è il suo profitto.



Esempio: enumerazione di tutte le soluzioni dell'RCP (4/5)

Gli schemi di taglio $(5), (6), (7), (8)$:



Esempio: enumerazione di tutte le soluzioni dell'RCP (5/5)

Etichetta	Vettore soluzione	Numero di pezzi	Profitto totale
(1)	$(0,0,0,1)$	1	9
(2)	$(1,0,1,0)$	2	$8+1=9$
(3)	$(0,2,0,0)$	2	$5+5=10$
(4)	$(2,1,0,0)$	3	$5+1+1=7$
(5)	$(1,0,1,0)$	2	$1+8=9$
(6)	$(2,1,0,0)$	3	$1+5+1=7$
(7)	$(2,1,0,0)$	3	$1+1+5=7$
(8)	$(4,0,0,0)$	4	$1+1+1+1=4$

Riepilogo

Istanze e soluzioni ammissibili:

- Un'istanza dell'RCP: un'asta di lunghezza n e un profitto p_ℓ per ogni lunghezza dei pezzi $\ell \in \{1, 2, \ldots, n\}$
- Un'asta di lunghezza n si può tagliare in $2^{n-1} = O(2^n)$ modi: troppi da enumerare

Codifica:

- Un vettore intero $(x_1, x_2, \ldots, x_n)$, con x_ℓ pezzi di lunghezza ℓ, $\sum_{\ell=1}^{n} \ell\, x_\ell = n$ e profitto $\sum_{\ell=1}^{n} p_\ell x_\ell$
- Schemi di taglio diversi possono avere lo stesso vettore, e allora lo stesso profitto totale

Un primo algoritmo di PD

Introduzione: Un primo algoritmo di PD

- Scriviamo una ricorrenza $v(\ell)$ sulle lunghezze dell'asta, e verifichiamo la sottostruttura ottima e i sottoproblemi sovrapposti
- Passo 1: calcoliamo il valore ottimo, top-down e bottom-up
- Passo 2: memorizziamo il primo pezzo di ogni lunghezza dell'asta e ricostruiamo una soluzione ottima
- Infine contiamo i passi dell'algoritmo di PD

Passo 1: calcolo del valore ottimo

Per ogni $\ell \in \{0, 1, \ldots, n\}$, sia $v(\ell)$ il valore ottimo della sottoistanza dell'RCP data da:

1. un'asta di lunghezza ℓ,
2. un insieme $\{1, 2, \ldots, \ell\}$ di ℓ possibili lunghezze dei pezzi,
3. per ogni lunghezza dei pezzi $j \in \{1, 2, \ldots, \ell\}$, un profitto p_j.

La formula ricorsiva per $v(\ell)$

La formula ricorsiva per calcolare $v(\ell)$ è la seguente:

$$v(\ell) = \begin{cases} \max\left\{p_j + v(\ell - j) : j \in \{1, 2, \ldots, \ell\}\right\} & \text{se } 1 \leq \ell \leq n, \\ 0 & \text{se } \ell = 0. \end{cases}$$

Sottoproblemi indipendenti e caso base

- Il processo di decomposizione ricorsiva della formula ricorsiva per $v(\ell)$ genera ℓ **sottoproblemi indipendenti**, ciascuno corrispondente a una possibile scelta di tagliare un primo pezzo di lunghezza j da un'asta di lunghezza ℓ.

- Qui $v(0) = 0$, perché un'asta di lunghezza 0 non si può tagliare e ha profitto 0.

Sottostruttura ottima

- Il valore ottimo $v(\ell)$ si ottiene provando tutte le possibili lunghezze del primo pezzo e scegliendo quella che massimizza la somma $p_j + v(\ell - j)$ del profitto del primo pezzo e del valore ottimo del sottoproblema residuo.

- Quindi una soluzione ottima per un'asta di lunghezza ℓ si costruisce combinando il primo pezzo scelto, di lunghezza j, con una soluzione ottima del sottoproblema residuo di lunghezza $\ell - j$.

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

Sottoproblemi sovrapposti

- Scelte diverse della lunghezza del primo pezzo, eventualmente fatte per valori diversi di ℓ, possono portare alla stessa lunghezza residua.
- Quindi lo stesso valore $v(\ell)$ può dover essere calcolato più volte durante la valutazione ricorsiva della formula ricorsiva per $v(\ell)$.

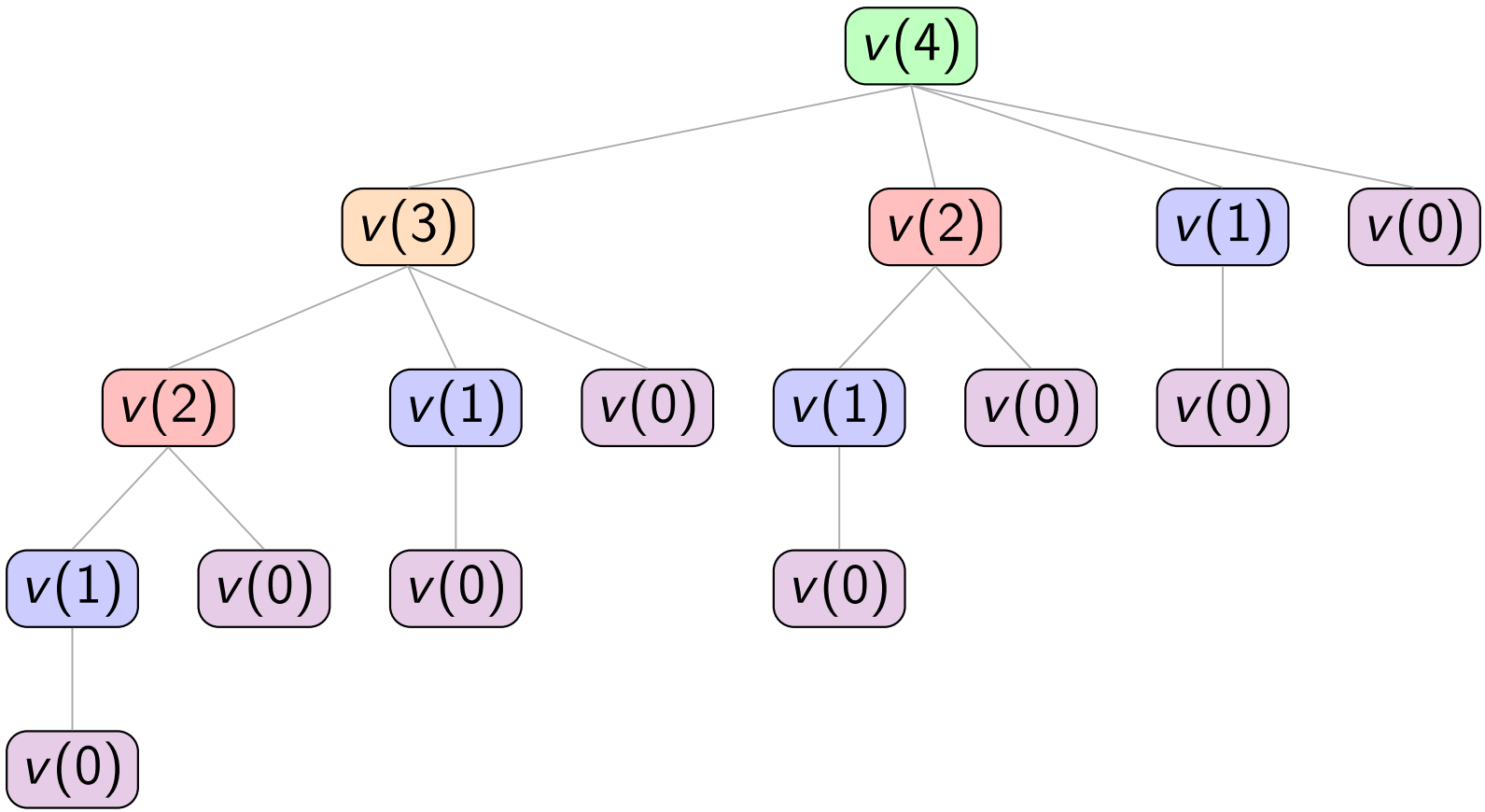
$\rightsquigarrow$ **Sottoproblemi sovrapposti verificati!**

L'approccio top-down

- Un **primo metodo** per calcolare il valore ottimo $v(n)$ dell'istanza dell'RCP è l'**approccio top-down**.
- Il metodo parte dalla chiamata con $\ell = n$ e valuta ricorsivamente la formula ricorsiva per $v(\ell)$: ogni chiamata per un'asta di lunghezza ℓ genera altre chiamate per aste di lunghezza $\ell - j$, per ogni $j \in \{1, 2, \ldots, \ell\}$, fino a raggiungere il caso base $v(0)$.
- 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 fatte da quella chiamata. Le foglie rappresentano i casi base.

Esempio: albero di ricorsione (1/2)

Consideriamo l'istanza di riferimento. Questo è l'albero di ricorsione dell'approccio top-down per calcolare $v(4)$:



Esempio: albero di ricorsione (2/2)

La funzione v viene chiamata 16 volte, e gli stessi sottoproblemi vengono risolti più volte:

- $v(2)$ viene chiamata 2 volte,
- $v(1)$ viene chiamata 4 volte,
- $v(0)$ viene chiamata 8 volte.

Tutte le altre chiamate vengono fatte una sola volta.

Calcolare i valori per lunghezze crescenti

Il valore $v(\ell)$, per una generica lunghezza dell'asta ℓ,
si può calcolare una volta noti i valori $v(j)$
per tutte le lunghezze minori $j < \ell$.

Esempio: calcolo dei valori della funzione v (1/2)

Consideriamo l'istanza di riferimento, con un'asta di lunghezza $n = 4$ e i seguenti profitti delle possibili lunghezze dei pezzi:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
p_ℓ	1	5	8	9

Il valore della funzione v si può calcolare partendo dal caso base $v(0) = 0$ e calcolando poi, in ordine, i valori di $v(1)$, $v(2)$, $v(3)$ e $v(4)$, come segue:

- Poiché conosciamo il valore di $v(0)$, possiamo calcolare $v(1)$ come segue:

$$v(1) = p_1 + v(0) = 1 + 0 = 1$$

- Calcolato $v(1)$, possiamo calcolare $v(2)$ come segue:

$$v(2) = \max\{p_1 + v(1), p_2 + v(0)\} = \max\{1 + 1, 5 + 0\} = 5$$

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

- Calcolati $v(2)$ e $v(1)$, possiamo calcolare $v(3)$ come segue:

$$\begin{aligned} v(3) &= \max\{p_1 + v(2), p_2 + v(1), p_3 + v(0)\} \\ &= \max\{1 + 5, 5 + 1, 8 + 0\} = 8 \end{aligned}$$

- Calcolati $v(3)$, $v(2)$ e $v(1)$, possiamo calcolare $v(4)$ come segue:

$$\begin{aligned} v(4) &= \max\{p_1 + v(3), p_2 + v(2), p_3 + v(1), p_4 + v(0)\} \\ &= \max\{1 + 8, 5 + 5, 8 + 1, 9 + 0\} = 10 \end{aligned}$$

La tabella di lookup

- Possiamo evitare di risolvere ripetutamente gli stessi sottoproblemi memorizzando il valore di ogni sottoproblema appena calcolato. I valori si memorizzano in una **tabella di lookup**.

- Poiché la soluzione di un sottoproblema di valore $v(\ell)$ dipende solo dai sottoproblemi di valore $v(j)$, con $j < \ell$, possiamo risolvere i sottoproblemi per valori crescenti di ℓ, partendo dal caso base $v(0) = 0$ fino ad arrivare a $v(n)$.

- Questa osservazione porta in modo naturale a un **metodo molto più efficiente** per calcolare il valore ottimo $v(n)$ dell'istanza dell'RCP.

L'approccio bottom-up

- Un **secondo metodo** per calcolare il valore ottimo $v(n)$ dell'istanza dell'RCP è l'**approccio bottom-up**.
- Il metodo parte dal caso base $v(0) = 0$ e valuta la formula ricorsiva per $v(\ell)$ per lunghezze crescenti dell'asta $\ell \in \{1, 2, \ldots, n\}$: ogni valore $v(\ell)$ si calcola con i valori già calcolati $v(0), v(1), \ldots, v(\ell - 1)$, fino a ottenere il valore $v(n)$ dell'istanza dell'RCP.

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

Algorithm: DP1

Input : Un'istanza dell'RCP data dalla lunghezza n dell'asta e da un array $p[1..n]$ dei profitti dei pezzi

Output : Il valore ottimo dell'istanza dell'RCP

```
1  function DP1(n, p)
2      v := new array[0..n]
3      v[0] ← 0
4      for ℓ := 1 to n do
5          m := 0
6          for j := 1 to ℓ do
7              if m < p[j] + v[ℓ − j] then
8                  m ← p[j] + v[ℓ − j]
9              end
10         end
11         v[ℓ] ← m
12     end
13     return v[n]
14 end
```

Come funziona l'algoritmo di PD

- Il vettore $v[0..n]$ è la **tabella di lookup**: memorizza i valori ottimi di tutti i sottoproblemi, con il caso base $v[0] = 0$.
- Il ciclo esterno, su $\ell \in \{1, 2, \ldots, n\}$, calcola $v[\ell]$ in ordine crescente di lunghezza dell'asta.
- Il ciclo interno, su $j \in \{1, 2, \ldots, \ell\}$, valuta tutte le possibili lunghezze del primo pezzo e tiene in m il valore massimo di $p[j] + v[\ell - j]$.
- Alla fine del ciclo interno, $v[\ell] \leftarrow m$ memorizza il valore ottimo $v[\ell]$.
- Infine, l'algoritmo restituisce il valore ottimo $v[n]$ dell'istanza dell'RCP.

Esempio: esecuzione dell'algoritmo di PD (1/4)

Consideriamo l'istanza di riferimento, con un'asta di lunghezza $n = 4$ e i seguenti profitti delle possibili lunghezze dei pezzi:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
p_ℓ	1	5	8	9

Per ogni passo dell'algoritmo di PD, le tabelle seguenti riportano:

- i valori di ℓ e j, la cella $v[\ell]$ e il valore corrente di m;
- la cella di m è evidenziata in rosso quando viene aggiornata;
- la cella di $v[\ell]$ è evidenziata in blu quando viene assegnata.

Esempio: esecuzione dell'algoritmo di PD (2/4)

Inizializzazione e lunghezze dell'asta $\ell = 1$ e $\ell = 2$:

ℓ	j	$v[\ell]$	m	Azione
–	–	0	–	$v[0] \leftarrow 0$
	–	–	0	$m \leftarrow 0$
1	1	–	1	$p[1]+v[0] = 1+0 = 1 > 0$: $m \leftarrow 1$
	–	1	1	$v[1] \leftarrow m = 1$
	–	–	0	$m \leftarrow 0$
2	1	–	2	$p[1]+v[1] = 1+1 = 2 > 0$: $m \leftarrow 2$
	2	–	5	$p[2]+v[0] = 5+0 = 5 > 2$: $m \leftarrow 5$
	–	5	5	$v[2] \leftarrow m = 5$

Esempio: esecuzione dell'algoritmo di PD (3/4)

Lunghezze dell'asta $\ell = 3$ e $\ell = 4$:

ℓ	j	$v[\ell]$	m	Azione
	–	–	0	$m \leftarrow 0$
3	1	–	6	$p[1]+v[2] = 1+5 = 6 > 0$: $m \leftarrow 6$
	2	–	6	$p[2]+v[1] = 5+1 = 6 = m$: nessun aggiornamento
	3	–	8	$p[3]+v[0] = 8+0 = 8 > 6$: $m \leftarrow 8$
	–	8	8	$v[3] \leftarrow m = 8$
	–	–	0	$m \leftarrow 0$
4	1	–	9	$p[1]+v[3] = 1+8 = 9 > 0$: $m \leftarrow 9$
	2	–	10	$p[2]+v[2] = 5+5 = 10 > 9$: $m \leftarrow 10$
	3	–	10	$p[3]+v[1] = 8+1 = 9 < 10$: nessun aggiornamento
	4	–	10	$p[4]+v[0] = 9+0 = 9 < 10$: nessun aggiornamento
	–	10	10	$v[4] \leftarrow m = 10$

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

Questo è il vettore v dopo l'esecuzione dell'algoritmo di PD:

	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$v[\ell]$	0	1	5	8	10

- Dopo l'esecuzione dell'algoritmo di PD, il vettore v contiene il valore ottimo $v[\ell]$ per ogni lunghezza dell'asta.
- Il valore ottimo dell'istanza di riferimento è $v[4] = 10$.

Passo 2: calcolo di una soluzione ottima

- Per calcolare una soluzione ottima è necessario memorizzare alcune informazioni aggiuntive.

- Per ogni lunghezza dell'asta ℓ, oltre al valore ottimo $v[\ell]$, memorizziamo $s[\ell]$, la lunghezza del primo pezzo da tagliare in una soluzione ottima per un'asta di lunghezza ℓ.

- Partendo da $\ell = n$, il valore $s[\ell]$ dà la lunghezza del primo pezzo da tagliare; l'asta rimanente ha lunghezza $\ell - s[\ell]$, e la stessa regola si può applicare di nuovo, e così via, finché la lunghezza residua arriva a 0.

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

Algorithm: DP1_InfoSol (1/2)

Input : Un'istanza dell'RCP data dalla lunghezza n dell'asta e da un array $p[1..n]$ dei profitti dei pezzi

Output: Il valore ottimo dell'istanza dell'RCP e il vettore di informazioni $s[1..n]$

```
1 function DP1_InfoSol(n, p)
2 |   v := new array[0..n]
3 |   s := new array[1..n]
4 |   v[0] ← 0
```

Algorithm: DP1_InfoSol (2/2)

function *DP1_InfoSol*(n, p) (continua)

```
5   |  for ℓ := 1 to n do
6   |  |  m := 0
7   |  |  t := 0
8   |  |  for j := 1 to ℓ do
9   |  |  |  if m < p[j] + v[ℓ − j] then
10  |  |  |  |  m ← p[j] + v[ℓ − j]
11  |  |  |  |  t ← j
12  |  |  |  end
13  |  |  end
14  |  |  v[ℓ] ← m
15  |  |  s[ℓ] ← t
16  |  end
17  |  return v[n], s
18  end
```

Come si memorizzano le informazioni

- L'algoritmo di PD usa la variabile temporanea aggiuntiva t per tenere traccia della decisione presa quando m viene aggiornato.
- Se si sceglie un primo pezzo di lunghezza j, $t \leftarrow j$ memorizza questa decisione.
- Alla fine del ciclo interno, $s[\ell] \leftarrow t$ memorizza la lunghezza del primo pezzo scelto, insieme al valore ottimo $v[\ell] \leftarrow m$.

Esempio: calcolo delle informazioni aggiuntive (1/4)

Consideriamo l'istanza di riferimento, con un'asta di lunghezza $n = 4$ e i seguenti profitti delle possibili lunghezze dei pezzi:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
p_ℓ	1	5	8	9

Per ogni passo dell'algoritmo di PD, le tabelle seguenti riportano:

- i valori di ℓ e j, la cella $s[\ell]$, il valore corrente di m e il valore corrente di t;
- la cella di m è evidenziata in rosso quando viene aggiornata;
- la cella di t e la cella di $s[\ell]$ sono evidenziate in blu quando vengono aggiornate.

Esempio: calcolo delle informazioni aggiuntive (2/4)

Stato iniziale e lunghezze dell'asta $\ell = 1$ e $\ell = 2$:

ℓ	j	$s[\ell]$	m	t	Azione
–	–	–	–	–	Stato iniziale
	–	–	0	0	$m \leftarrow 0,\ t \leftarrow 0$
1	1	–	1	1	$1{+}0 = 1 > 0$: $m \leftarrow 1,\ t \leftarrow 1$
	–	1	1	1	$v[1] \leftarrow m = 1,\ s[1] \leftarrow t = 1$
	–	–	0	0	$m \leftarrow 0,\ t \leftarrow 0$
2	1	–	2	1	$1{+}1 = 2 > 0$: $m \leftarrow 2,\ t \leftarrow 1$
	2	–	5	2	$5{+}0 = 5 > 2$: $m \leftarrow 5,\ t \leftarrow 2$
	–	2	5	2	$v[2] \leftarrow m = 5,\ s[2] \leftarrow t = 2$

Esempio: calcolo delle informazioni aggiuntive (3/4)

Lunghezze dell'asta $\ell = 3$ e $\ell = 4$:

ℓ	j	$s[\ell]$	m	t	Azione
	–	–	0	0	$m \leftarrow 0,\ t \leftarrow 0$
3	1	–	6	1	$1{+}5 = 6 > 0$: $m \leftarrow 6,\ t \leftarrow 1$
	2	–	6	1	$5{+}1 = 6 = m$: nessun aggiornamento
	3	–	8	3	$8{+}0 = 8 > 6$: $m \leftarrow 8,\ t \leftarrow 3$
	–	3	8	3	$v[3] \leftarrow m = 8,\ s[3] \leftarrow t = 3$
	–	–	0	0	$m \leftarrow 0,\ t \leftarrow 0$
4	1	–	9	1	$1{+}8 = 9 > 0$: $m \leftarrow 9,\ t \leftarrow 1$
	2	–	10	2	$5{+}5 = 10 > 9$: $m \leftarrow 10,\ t \leftarrow 2$
	3	–	10	2	$8{+}1 = 9 < 10$: nessun aggiornamento
	4	–	10	2	$9{+}0 = 9 < 10$: nessun aggiornamento
	–	2	10	2	$v[4] \leftarrow m = 10,\ s[4] \leftarrow t = 2$

Esempio: calcolo delle informazioni aggiuntive (4/4)

Questo è il vettore s dopo l'esecuzione dell'algoritmo di PD:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$s[\ell]$	1	2	3	2

- Per esempio, $s[4] = 2$: in una soluzione ottima per l'asta di lunghezza 4, il primo pezzo da tagliare ha lunghezza 2.

Calcolo di una soluzione ottima dal vettore s

- Calcolato il vettore s, si può calcolare una soluzione ottima dell'istanza dell'RCP applicando ripetutamente la seguente regola, partendo da $j = n$ e da $x_\ell = 0$ per ogni $\ell \in \{1, 2, \ldots, n\}$:

$$x_{s[j]} \leftarrow x_{s[j]} + 1, \qquad j \leftarrow j - s[j],$$

  finché $j = 0$.

- Si ottiene così il vettore $(x_1, x_2, \ldots, x_n)$, dove ogni componente x_ℓ è il numero di pezzi di lunghezza ℓ in una soluzione ottima.

- Il seguente pseudocodice implementa il calcolo di una soluzione ottima, dato il vettore s.

Algorithm: DP1_SolComputation

Input : La lunghezza n dell'asta dell'istanza dell'RCP e il vettore $s[1..n]$

Output: Una soluzione ottima dell'istanza dell'RCP

```
1  function DP1_SolComputation(n, s)
2      x := new array[1..n]
3      for ℓ := 1 to n do
4          x[ℓ] ← 0
5      end
6      j := n
7      while j > 0 do
8          x[s[j]] ← x[s[j]] + 1
9          j ← j − s[j]
10     end
11     return x
12 end
```

Esempio: calcolo di una soluzione ottima

Dopo l'esecuzione dell'algoritmo di PD sull'istanza di riferimento, abbiamo il seguente vettore s:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$s[\ell]$	1	2	3	2

Possiamo calcolare una soluzione ottima per un'asta di lunghezza $n = 4$ come segue:

Iterazione	j	$s[j]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$j \leftarrow j - s[j]$
–	–	–	0	0	0	0	$j = 4$
1	4	2	0	1	0	0	$j = 4 - 2 = 2$
2	2	2	0	2	0	0	$j = 2 - 2 = 0$

Una soluzione ottima è il vettore:

$$(0, 2, 0, 0)$$

Soluzioni ottime per ogni lunghezza dell'asta

La stessa procedura di calcolo, basata sul vettore s,
si può applicare partendo da qualunque lunghezza dell'asta ℓ.

- Il vettore s si calcola una volta sola, con l'algoritmo di PD.
- Dà una soluzione ottima per ogni lunghezza dell'asta $\ell \in \{1, 2, \ldots, n\}$.

Esempio: soluzioni ottime per le lunghezze 3 e 2 (1/2)

Dopo l'esecuzione dell'algoritmo di PD sull'istanza di riferimento, abbiamo il seguente vettore s:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$s[\ell]$	1	2	3	2

Possiamo calcolare una soluzione ottima per un'asta di lunghezza 3 come segue:

Iterazione	j	$s[j]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$j \leftarrow j - s[j]$
–	–	–	0	0	0	0	$j = 3$
1	3	3	0	0	1	0	$j = 3 - 3 = 0$

Una soluzione ottima è il vettore:

$$(0, 0, 1, 0)$$

Esempio: soluzioni ottime per le lunghezze 3 e 2 (2/2)

Possiamo calcolare una soluzione ottima per un'asta di lunghezza 2 come segue:

Iterazione	j	$s[j]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$j \leftarrow j - s[j]$
–	–	–	0	0	0	0	$j = 2$
1	2	2	0	1	0	0	$j = 2 - 2 = 0$

Una soluzione ottima è il vettore:

$$(0, 1, 0, 0)$$

Numero di passi

- Sia $S(n)$ il numero di passi eseguiti da `DP1` nel caso peggiore, dove n è la lunghezza della asta.
- Calcoliamo $S(n)$ contando quante volte vengono eseguite le Righe 2–13.
- Nel caso peggiore, la condizione alla riga 7 è vera a ogni iterazione del ciclo interno, e quindi la riga 8 viene eseguita tante volte quanto la riga 7.

Algorithm: DP1

Input : Un'istanza dell'RCP data dalla lunghezza n dell'asta e da un array $p[1..n]$ dei profitti dei pezzi

Output : Il valore ottimo dell'istanza dell'RCP

```
1  function DP1(n, p)
2      v := new array[0..n]
3      v[0] ← 0
4      for ℓ := 1 to n do
5          m := 0
6          for j := 1 to ℓ do
7              if m < p[j] + v[ℓ − j] then
8                  m ← p[j] + v[ℓ − j]
9              end
10         end
11         v[ℓ] ← m
12     end
13     return v[n]
14 end
```

Numero di esecuzioni delle righe

Il numero di esecuzioni delle righe rilevanti è:

- Riga 2 viene eseguita 1 volta.
- Riga 3 viene eseguita 1 volta.
- Riga 4 viene eseguita n volte.
- Riga 5 viene eseguita n volte.
- Riga 6 viene eseguita $\sum_{\ell=1}^{n} \ell$ volte.
- Riga 7 viene eseguita $\sum_{\ell=1}^{n} \ell$ volte.
- Riga 8 viene eseguita $\sum_{\ell=1}^{n} \ell$ volte.
- Riga 11 viene eseguita n volte.
- Riga 13 viene eseguita 1 volta.

Il numero totale di passi

Quindi il numero totale di passi nel caso peggiore è

$$1 + 1 + n + n + \sum_{\ell=1}^{n} \ell + \sum_{\ell=1}^{n} \ell + \sum_{\ell=1}^{n} \ell + n + 1 = 3 + 3n + 3 \sum_{\ell=1}^{n} \ell.$$

Usando $\sum_{\ell=1}^{n} \ell = \dfrac{n(n+1)}{2}$, otteniamo

$$S(n) = 3 + 3n + 3 \cdot \frac{n(n+1)}{2} = \frac{3n^2 + 9n + 6}{2}.$$

Quindi

$$S(n) = \Theta(n^2).$$

Il numero di passi è quadratico

Infatti,

$$\lim_{n\to\infty} \frac{\frac{3n^2+9n+6}{2}}{n^2} = \lim_{n\to\infty} \frac{1}{2}\left(3 + \frac{9}{n} + \frac{6}{n^2}\right) = \frac{3}{2} \in \mathbb{R}_{>0}.$$

L'algoritmo di PD esegue

$$S(n) = \Theta(n^2)$$

passi nel caso peggiore.

Riepilogo (1/2)

Passo 1: il valore ottimo:

- $v(\ell) = \max\{p_j + v(\ell - j) : j \in \{1, 2, \ldots, \ell\}\}$ per $1 \leq \ell \leq n$, con il caso base $v(0) = 0$
- sottostruttura ottima e sottoproblemi sovrapposti: l'albero di ricorsione top-down per $v(4)$ ha 16 chiamate
- bottom-up: il vettore v si riempie per ℓ crescente; sull'istanza di riferimento $v[4] = 10$

Passo 2: una soluzione ottima:

- $s[\ell]$ è la lunghezza del primo pezzo da tagliare da un'asta di lunghezza ℓ
- regola $x_{s[j]} \leftarrow x_{s[j]} + 1$, $j \leftarrow j - s[j]$ finché $j = 0$

Riepilogo (2/2)

Soluzioni sull'istanza di riferimento:

- $s = (1, 2, 3, 2)$
- asta di lunghezza 4: $(0, 2, 0, 0)$, due pezzi di lunghezza 2, profitto 10
- aste di lunghezza 3 e 2: $(0, 0, 1, 0)$ e $(0, 1, 0, 0)$

Numero di passi:

- `DP1` esegue $S(n) = \dfrac{3n^2 + 9n + 6}{2} = \Theta(n^2)$ passi nel caso peggiore

Un secondo algoritmo di PD

Introduzione: Un secondo algoritmo di PD

- Scriviamo una seconda ricorrenza $v(j, \ell)$, sulle coppie (lunghezza massima dei pezzi, lunghezza della asta), come nell'algoritmo di PD del problema dello zaino 0/1
- Per ogni lunghezza dei pezzi j decidiamo se tagliare o no un pezzo di lunghezza j: al più 2 sottoproblemi invece di ℓ
- Seguiamo gli stessi passi: valore ottimo top-down e bottom-up, soluzione ottima, numero di passi

Passo 1: calcolo del valore ottimo

Per ogni $j \in \{0, 1, \ldots, n\}$ e $\ell \in \{0, 1, \ldots, n\}$, sia $v(j, \ell)$ il valore ottimo della sottoistanza dell'RCP data da:

1. un'asta di lunghezza ℓ,
2. un insieme $\{1, 2, \ldots, j\}$ di j possibili lunghezze dei pezzi,
3. per ogni lunghezza dei pezzi $k \in \{1, 2, \ldots, j\}$, un profitto p_k.

La formula ricorsiva per $v(j, \ell)$

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

$$v(j,\ell) = \begin{cases} \max\left\{v(j-1,\ell),\ p_j + v(j,\ell-j)\right\} & \text{se } 1 \le j \le \ell, \\ v(j-1,\ell) & \text{se } 1 \le \ell < j, \\ 0 & \text{se } \ell = 0, \\ -\infty & \text{se } j = 0 \text{ e } \ell > 0. \end{cases}$$

Sottoproblemi indipendenti e casi base

- Il processo di decomposizione ricorsiva della formula ricorsiva per $v(j, \ell)$ genera al più 2 **sottoproblemi indipendenti**, corrispondenti alla scelta di tagliare o no un pezzo di lunghezza j da un'asta di lunghezza ℓ. Se $j > \ell$, un pezzo di lunghezza j non si può tagliare dall'asta, e quindi l'unica opzione ammissibile è non tagliarlo.

- Qui $v(j, 0) = 0$, perché un'asta di lunghezza 0 non si può tagliare e ha profitto 0.

- Inoltre, $v(0, \ell) = -\infty$ per $\ell > 0$: in questo caso non è disponibile alcuna lunghezza positiva dei pezzi, quindi non esiste una soluzione ammissibile per un'asta di lunghezza positiva. Il valore $-\infty$ serve a escludere questo caso inammissibile.

Sottostruttura ottima

- Se $1 \leq j \leq \ell$, il valore ottimo $v(j, \ell)$ si ottiene provando le due opzioni possibili, tagliare o non tagliare un primo pezzo di lunghezza j, e scegliendo quella che massimizza il profitto. I valori corrispondenti sono rispettivamente $p_j + v(j, \ell - j)$ e $v(j - 1, \ell)$.
- Se $1 \leq \ell < j$, la lunghezza j non si può usare, e quindi $v(j, \ell) = v(j - 1, \ell)$.
- Quindi una soluzione ottima per un'asta di lunghezza ℓ, con lunghezza massima disponibile dei pezzi j, si costruisce combinando la decisione presa sulla lunghezza j con una soluzione ottima del sottoproblema residuo corrispondente.

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

Sottoproblemi sovrapposti

- Sequenze diverse di decisioni sulle lunghezze dei pezzi, eventualmente prese per valori diversi di j e ℓ, possono portare allo stesso sottoproblema.
- Quindi lo stesso valore $v(j, \ell)$ può dover essere calcolato più volte durante la valutazione ricorsiva della formula ricorsiva per $v(j, \ell)$.

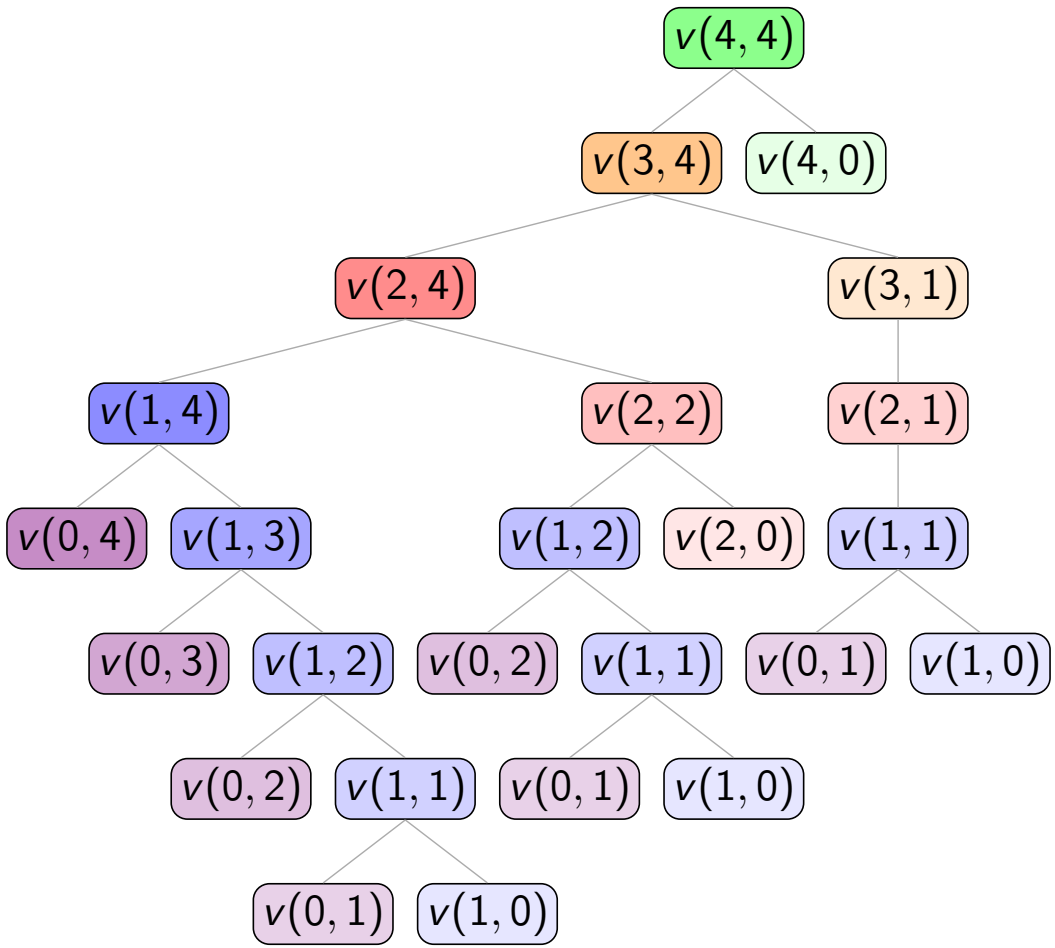
$\rightsquigarrow$ **Sottoproblemi sovrapposti verificati!**

L'approccio top-down

- Un **primo metodo** per calcolare il valore ottimo $v(n, n)$ dell'istanza dell'RCP è l'**approccio top-down**.
- Il metodo parte dalla chiamata con $(j, \ell) = (n, n)$ e valuta ricorsivamente la formula ricorsiva per $v(j, \ell)$: ogni chiamata per un'asta di lunghezza ℓ e lunghezza massima disponibile dei pezzi j genera altre chiamate, corrispondenti alle decisioni ammissibili sulla lunghezza j, fino a raggiungere uno dei casi base $v(j, 0)$ o $v(0, \ell)$.
- 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 fatte da quella chiamata. Le foglie rappresentano i casi base.

Esempio: albero di ricorsione (1/2)

Consideriamo l'istanza di riferimento. Questo è l'albero di ricorsione dell'approccio top-down per calcolare $v(4, 4)$:



Esempio: albero di ricorsione (2/2)

La funzione v viene chiamata 25 volte, e gli stessi sottoproblemi vengono risolti più volte:

- $v(1, 2)$ viene chiamata 2 volte,
- $v(0, 2)$ viene chiamata 2 volte,
- $v(1, 1)$ viene chiamata 3 volte,
- $v(0, 1)$ viene chiamata 3 volte,
- $v(1, 0)$ viene chiamata 3 volte.

Tutte le altre chiamate vengono fatte una sola volta.

Calcolare i valori riga per riga

- Il valore $v(j, \ell)$ si può calcolare una volta noto $v(j-1, \ell)$ e, quando $j \leq \ell$, una volta noto anche $v(j, \ell - j)$.
- Quindi, per un valore fissato di j, tutti i valori $v(j, \ell)$ si possono calcolare considerando i valori di ℓ in ordine crescente.
- Partendo dai casi base $v(j, 0) = 0$ e $v(0, \ell) = -\infty$, i valori della funzione v si possono calcolare riga per riga, passando al valore successivo di j solo dopo aver calcolato tutti i valori $v(j, \ell)$ per il valore corrente di j.

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

Consideriamo l'istanza di riferimento, con i profitti $p_1 = 1$, $p_2 = 5$, $p_3 = 8$, $p_4 = 9$. I valori della funzione v si calcolano come segue.

Per $j = 1$ si possono usare solo pezzi di lunghezza 1. Quindi:

$$v(1,0) = 0,$$

$$v(1,1) = \max\{v(0,1), p_1 + v(1,0)\} = \max\{-\infty, 1+0\} = 1,$$

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

$$v(1,3) = \max\{v(0,3), p_1 + v(1,2)\} = \max\{-\infty, 1+2\} = 3,$$

$$v(1,4) = \max\{v(0,4), p_1 + v(1,3)\} = \max\{-\infty, 1+3\} = 4.$$

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

Per $j = 2$ si possono usare pezzi di lunghezza 1 e 2. Otteniamo:

$$v(2,0) = 0, \qquad v(2,1) = v(1,1) = 1,$$

$$v(2,2) = \max\{v(1,2), p_2 + v(2,0)\} = \max\{2, 5+0\} = 5,$$

$$v(2,3) = \max\{v(1,3), p_2 + v(2,1)\} = \max\{3, 5+1\} = 6,$$

$$v(2,4) = \max\{v(1,4), p_2 + v(2,2)\} = \max\{4, 5+5\} = 10.$$

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

Per $j = 3$ si possono usare pezzi di lunghezza 1, 2 e 3. Otteniamo:

$$v(3,0) = 0, \qquad v(3,1) = v(2,1) = 1, \qquad v(3,2) = v(2,2) = 5,$$

$$v(3,3) = \max\{v(2,3), p_3 + v(3,0)\} = \max\{6, 8+0\} = 8,$$

$$v(3,4) = \max\{v(2,4), p_3 + v(3,1)\} = \max\{10, 8+1\} = 10.$$

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

Per $j = 4$ sono disponibili tutte le lunghezze dei pezzi. Otteniamo:

$$v(4,0) = 0, \qquad v(4,1) = v(3,1) = 1,$$

$$v(4,2) = v(3,2) = 5, \qquad v(4,3) = v(3,3) = 8,$$

$$v(4,4) = \max\{v(3,4), p_4 + v(4,0)\} = \max\{10, 9+0\} = 10.$$

La tabella di lookup

- Possiamo evitare di risolvere ripetutamente gli stessi sottoproblemi memorizzando il valore di ogni sottoproblema appena calcolato. I valori si memorizzano in una **tabella di lookup**.

- Poiché la soluzione di ogni sottoproblema dipende solo dai sottoproblemi con un valore minore di j, o con lo stesso valore di j e una lunghezza dell'asta minore, possiamo risolvere i sottoproblemi in ordine crescente di j e, per ogni valore fissato di j, in ordine crescente di lunghezza dell'asta, partendo dai casi base $v(0,0) = 0$ e $v(0,\ell) = -\infty$ per ogni $\ell \in \{1,\ldots,n\}$ fino ad arrivare a $v(n,n)$.

- Questa osservazione porta in modo naturale a un **metodo molto più efficiente** per calcolare il valore ottimo $v(n,n)$ dell'istanza dell'RCP.

L'approccio bottom-up

- Un **secondo metodo** per calcolare il valore ottimo $v(n, n)$ dell'istanza dell'RCP è l'**approccio bottom-up**.

- Il metodo parte dai casi base $v(0, 0) = 0$ e $v(0, \ell) = -\infty$ per ogni $\ell \in \{1, \ldots, n\}$ e valuta la formula ricorsiva per $v(j, \ell)$ per valori crescenti di $j \in \{1, 2, \ldots, n\}$ e $\ell \in \{0, 1, \ldots, n\}$: ogni valore $v(j, \ell)$ si calcola con valori già calcolati, fino a ottenere il valore $v(n, n)$ dell'istanza dell'RCP.

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

Algorithm: DP2 (1/2)

Input : Un'istanza dell'RCP data dalla lunghezza n dell'asta e da un array $p[1..n]$ dei profitti dei pezzi

Output : Il valore ottimo dell'istanza dell'RCP

```
1 function DP2(n, p)
2 |   v := new array[0..n][0..n]
3 |   v[0][0] ← 0
4 |   for ℓ := 1 to n do
5 |   |   v[0][ℓ] ← −∞
6 |   end
```

Algorithm: DP2 (2/2)

```
function DP2(n, p) (continua)
7     for j := 1 to n do
8         for ℓ := 0 to n do
9             if j > ℓ then
10                v[j][ℓ] ← v[j − 1][ℓ]
11            else
12                if p[j] + v[j][ℓ − j] > v[j − 1][ℓ] then
13                    v[j][ℓ] ← p[j] + v[j][ℓ − j]
14                else
15                    v[j][ℓ] ← v[j − 1][ℓ]
16                end
17            end
18        end
19    end
20    return v[n][n]
21 end
```

Come funziona l'algoritmo di PD (1/2)

- La matrice $v[0..n][0..n]$ è la **tabella di lookup**: memorizza i valori ottimi di tutti i sottoproblemi.
- L'inizializzazione pone $v[0][0] = 0$ e $v[0][\ell] = -\infty$ per ogni lunghezza dell'asta $\ell \in \{1, \ldots, n\}$, perché una lunghezza positiva non si può riempire se non è disponibile alcuna lunghezza dei pezzi.
- Il ciclo esterno, su $j \in \{1, 2, \ldots, n\}$, considera le lunghezze dei pezzi disponibili in ordine crescente.
- Il ciclo interno, su $\ell \in \{0, 1, \ldots, n\}$, calcola il valore migliore per ogni lunghezza della asta usando solo pezzi di lunghezza al più j.

Come funziona l'algoritmo di PD (2/2)

- Se $j > \ell$, la lunghezza j non si può usare, e quindi $v[j][\ell] \leftarrow v[j-1][\ell]$.
- Altrimenti, l'algoritmo di PD confronta il valore ottenuto senza usare la lunghezza j, cioè $v[j-1][\ell]$, con il valore ottenuto usando un pezzo di lunghezza j, cioè $p[j] + v[j][\ell - j]$.
- Infine, l'algoritmo restituisce il valore ottimo dell'istanza dell'RCP, $v[n][n]$.

Esempio: esecuzione dell'algoritmo di PD (1/5)

Consideriamo l'istanza di riferimento, con un'asta di lunghezza $n = 4$ e i seguenti profitti delle possibili lunghezze dei pezzi:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
p_ℓ	1	5	8	9

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

Esempio: esecuzione dell'algoritmo di PD (2/5)

Righe $j = 0$ e $j = 1$:

j	ℓ	$v[j][\ell]$	Azione
0	0	0	$v[0][0] \leftarrow 0$
0	1	$-\infty$	$v[0][1] \leftarrow -\infty$
0	2	$-\infty$	$v[0][2] \leftarrow -\infty$
0	3	$-\infty$	$v[0][3] \leftarrow -\infty$
0	4	$-\infty$	$v[0][4] \leftarrow -\infty$
1	0	0	$v[1][0] \leftarrow v[0][0] = 0$
1	1	1	$\max\{v[0][1], p[1] + v[1][0]\} = \max\{-\infty, 1 + 0\} = 1$
1	2	2	$\max\{v[0][2], p[1] + v[1][1]\} = \max\{-\infty, 1 + 1\} = 2$
1	3	3	$\max\{v[0][3], p[1] + v[1][2]\} = \max\{-\infty, 1 + 2\} = 3$
1	4	4	$\max\{v[0][4], p[1] + v[1][3]\} = \max\{-\infty, 1 + 3\} = 4$

Esempio: esecuzione dell'algoritmo di PD (3/5)

Righe $j = 2$ e $j = 3$:

j	ℓ	$v[j][\ell]$	Azione
2	0	0	$v[2][0] \leftarrow v[1][0] = 0$
2	1	1	$2 > 1$: $v[2][1] \leftarrow v[1][1] = 1$
2	2	5	$\max\{v[1][2], p[2] + v[2][0]\} = \max\{2, 5 + 0\} = 5$
2	3	6	$\max\{v[1][3], p[2] + v[2][1]\} = \max\{3, 5 + 1\} = 6$
2	4	10	$\max\{v[1][4], p[2] + v[2][2]\} = \max\{4, 5 + 5\} = 10$
3	0	0	$v[3][0] \leftarrow v[2][0] = 0$
3	1	1	$3 > 1$: $v[3][1] \leftarrow v[2][1] = 1$
3	2	5	$3 > 2$: $v[3][2] \leftarrow v[2][2] = 5$
3	3	8	$\max\{v[2][3], p[3] + v[3][0]\} = \max\{6, 8 + 0\} = 8$
3	4	10	$\max\{v[2][4], p[3] + v[3][1]\} = \max\{10, 8 + 1\} = 10$

Esempio: esecuzione dell'algoritmo di PD (4/5)

Riga $j = 4$:

j	ℓ	$v[j][\ell]$	Azione
4	0	0	$v[4][0] \leftarrow v[3][0] = 0$
4	1	1	$4 > 1$: $v[4][1] \leftarrow v[3][1] = 1$
4	2	5	$4 > 2$: $v[4][2] \leftarrow v[3][2] = 5$
4	3	8	$4 > 3$: $v[4][3] \leftarrow v[3][3] = 8$
4	4	10	$\max\{v[3][4], p[4] + v[4][0]\} = \max\{10, 9 + 0\} = 10$

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

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

$v[j][\ell]$	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$j = 0$	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$j = 1$	0	1	2	3	4
$j = 2$	0	1	5	6	10
$j = 3$	0	1	5	8	10
$j = 4$	0	1	5	8	10

- Dopo l'esecuzione dell'algoritmo di PD, la matrice v contiene il valore ottimo $v[j][\ell]$ per ogni sottoproblema (j, ℓ).
- Il valore ottimo dell'istanza di riferimento è $v[4][4] = 10$.

Passo 2: calcolo di una soluzione ottima

- Per calcolare una soluzione ottima è necessario memorizzare alcune informazioni aggiuntive.
- Per ogni valore j della lunghezza massima disponibile dei pezzi e per ogni lunghezza dell'asta ℓ, oltre al valore ottimo $v[j][\ell]$, memorizziamo $s[j][\ell]$, che vale 1 se si usa un pezzo di lunghezza j e 0 altrimenti.
- Partendo da $j = n$ e $\ell = n$, se $s[j][\ell] = 0$ la lunghezza j non viene usata e j diminuisce di 1. Se $s[j][\ell] = 1$, si usa un pezzo di lunghezza j e la lunghezza residua dell'asta diventa $\ell - j$. La procedura si ferma quando la lunghezza residua arriva a 0.

Algorithm: DP2_InfoSol (1/3)

Input : Un'istanza dell'RCP data dalla lunghezza n dell'asta e da un array $p[1..n]$ dei profitti dei pezzi

Output : Il valore ottimo dell'istanza dell'RCP e la matrice di informazioni binarie $s[0..n][0..n]$

```
1 function DP2_InfoSol(n, p)
2     v := new array[0..n][0..n]
3     s := new array[0..n][0..n]
4     v[0][0] ← 0
5     s[0][0] ← 0
6     for ℓ := 1 to n do
7         v[0][ℓ] ← −∞
8         s[0][ℓ] ← 0
9     end
```

Algorithm: DP2_InfoSol (2/3)

function *DP2_InfoSol*(n, p) (continua)

```
10  |  for j := 1 to n do
11  |  |  for ℓ := 0 to n do
12  |  |  |  if j > ℓ then
13  |  |  |  |  v[j][ℓ] ← v[j − 1][ℓ]
14  |  |  |  |  s[j][ℓ] ← 0
```

Algorithm: DP2_InfoSol (3/3)

```
function DP2_InfoSol(n, p) (continua)
    for j := 1 to n (continua) do
        for ℓ := 0 to n (continua) do
            else
                if p[j] + v[j][ℓ − j] > v[j − 1][ℓ] then
                    v[j][ℓ] ← p[j] + v[j][ℓ − j]
                    s[j][ℓ] ← 1
                else
                    v[j][ℓ] ← v[j − 1][ℓ]
                    s[j][ℓ] ← 0
                end
            end
        end
    end
    return v[n][n], s
end
```

Come si memorizzano le informazioni

- L'algoritmo di PD usa la matrice aggiuntiva s, che tiene traccia della decisione presa quando si calcola $v[j][\ell]$.
- Se si sceglie un pezzo di lunghezza j, $s[j][\ell] \leftarrow 1$ memorizza questa decisione.
- Altrimenti $s[j][\ell] \leftarrow 0$ registra che la lunghezza j non viene usata.

Esempio: calcolo delle informazioni aggiuntive (1/5)

Consideriamo l'istanza di riferimento, con un'asta di lunghezza $n = 4$ e i seguenti profitti delle possibili lunghezze dei pezzi:

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
p_ℓ	1	5	8	9

Per ogni passo dell'algoritmo di PD, le tabelle seguenti riportano i valori di j e ℓ, la cella $s[j][\ell]$ e l'operazione eseguita.

Esempio: calcolo delle informazioni aggiuntive (2/5)

Righe $j = 0$ e $j = 1$:

j	ℓ	$s[j][\ell]$	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	$s[0][4] \leftarrow 0$
1	0	0	$1 > 0$: $s[1][0] \leftarrow 0$
1	1	1	$p[1] + v[1][0] = 1 + 0 > v[0][1] = -\infty$: $s[1][1] \leftarrow 1$
1	2	1	$p[1] + v[1][1] = 1 + 1 > v[0][2] = -\infty$: $s[1][2] \leftarrow 1$
1	3	1	$p[1] + v[1][2] = 1 + 2 > v[0][3] = -\infty$: $s[1][3] \leftarrow 1$
1	4	1	$p[1] + v[1][3] = 1 + 3 > v[0][4] = -\infty$: $s[1][4] \leftarrow 1$

Esempio: calcolo delle informazioni aggiuntive (3/5)

Righe $j = 2$ e $j = 3$:

j	ℓ	$s[j][\ell]$	Azione
2	0	0	$2 > 0$: $s[2][0] \leftarrow 0$
2	1	0	$2 > 1$: $s[2][1] \leftarrow 0$
2	2	1	$p[2] + v[2][0] = 5 + 0 > v[1][2] = 2$: $s[2][2] \leftarrow 1$
2	3	1	$p[2] + v[2][1] = 5 + 1 > v[1][3] = 3$: $s[2][3] \leftarrow 1$
2	4	1	$p[2] + v[2][2] = 5 + 5 > v[1][4] = 4$: $s[2][4] \leftarrow 1$
3	0	0	$3 > 0$: $s[3][0] \leftarrow 0$
3	1	0	$3 > 1$: $s[3][1] \leftarrow 0$
3	2	0	$3 > 2$: $s[3][2] \leftarrow 0$
3	3	1	$p[3] + v[3][0] = 8 + 0 > v[2][3] = 6$: $s[3][3] \leftarrow 1$
3	4	0	$p[3] + v[3][1] = 8 + 1 \leq v[2][4] = 10$: $s[3][4] \leftarrow 0$

Esempio: calcolo delle informazioni aggiuntive (4/5)

Riga $j = 4$:

j	ℓ	$s[j][\ell]$	Azione
4	0	0	$4 > 0$: $s[4][0] \leftarrow 0$
4	1	0	$4 > 1$: $s[4][1] \leftarrow 0$
4	2	0	$4 > 2$: $s[4][2] \leftarrow 0$
4	3	0	$4 > 3$: $s[4][3] \leftarrow 0$
4	4	0	$p[4] + v[4][0] = 9 + 0 \leq v[3][4] = 10$: $s[4][4] \leftarrow 0$

Esempio: calcolo delle informazioni aggiuntive (5/5)

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

$s[j][\ell]$	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$j = 0$	0	0	0	0	0
$j = 1$	0	1	1	1	1
$j = 2$	0	0	1	1	1
$j = 3$	0	0	0	1	0
$j = 4$	0	0	0	0	0

Calcolo di una soluzione ottima dalla matrice s

- Calcolata la matrice s, si può calcolare una soluzione ottima per un'asta di lunghezza n applicando ripetutamente la seguente regola, partendo da $(j, \ell) = (n, n)$ e da $x_h = 0$ per ogni $h \in \{1, 2, \ldots, n\}$:

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

- La procedura si ferma quando $\ell = 0$. Si ottiene così il vettore $(x_1, x_2, \ldots, x_n)$, dove ogni componente x_h è il numero di pezzi di lunghezza h in una soluzione ottima.

- Il seguente pseudocodice implementa il calcolo di una soluzione ottima, data la matrice s.

Algorithm: DP2_SolComputation (1/2)

Input : La lunghezza n dell'asta e la matrice $s[0..n][0..n]$

Output: Una soluzione ottima dell'istanza dell'RCP

```
1 function DP2_SolComputation(n, s)
2 |   x := new array[1..n]
3 |   for h := 1 to n do
4 |   |   x[h] ← 0
5 |   end
6 |   j := n
7 |   ℓ := n
```

Algorithm: DP2_SolComputation (2/2)

function *DP2_SolComputation*(n, s) (continua)

```
 8   while ℓ > 0 do
 9       if s[j][ℓ] = 0 then
10           j ← j − 1
11       else
12           x[j] ← x[j] + 1
13           ℓ ← ℓ − j
14       end
15   end
16   return x
17 end
```

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

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

$s[j][\ell]$	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$j = 0$	0	0	0	0	0
$j = 1$	0	1	1	1	1
$j = 2$	0	0	1	1	1
$j = 3$	0	0	0	1	0
$j = 4$	0	0	0	0	0

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

Possiamo calcolare una soluzione ottima per un'asta di lunghezza $n = 4$ come segue:

Iterazione	j	ℓ	$s[j][\ell]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	Azione
1	4	4	0	0	0	0	0	$j \leftarrow 3$
2	3	4	0	0	0	0	0	$j \leftarrow 2$
3	2	4	1	0	1	0	0	$x[2] \leftarrow 1$, $\ell \leftarrow 4 - 2 = 2$
4	2	2	1	0	2	0	0	$x[2] \leftarrow 2$, $\ell \leftarrow 2 - 2 = 0$

Una soluzione ottima è il vettore:

$$(0, 2, 0, 0)$$

Soluzioni ottime per ogni lunghezza dell'asta

La stessa procedura di calcolo, basata sulla matrice s, si può applicare a qualunque lunghezza dell'asta $\ell \leq n$ partendo da $j = n$.

Esempio: calcolo di soluzioni ottime (1/3)

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

$s[j][\ell]$	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
$j = 0$	0	0	0	0	0
$j = 1$	0	1	1	1	1
$j = 2$	0	0	1	1	1
$j = 3$	0	0	0	1	0
$j = 4$	0	0	0	0	0

Esempio: calcolo di soluzioni ottime (2/3)

Possiamo calcolare una soluzione ottima per un'asta di lunghezza $\ell = 3$ come segue:

Iterazione	j	ℓ	$s[j][\ell]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	Azione
1	4	3	0	0	0	0	0	$j \leftarrow 3$
2	3	3	1	0	0	1	0	$x[3] \leftarrow 1$, $\ell \leftarrow 3 - 3 = 0$

Una soluzione ottima è il vettore:

$$(0, 0, 1, 0)$$

Esempio: calcolo di soluzioni ottime (3/3)

Possiamo calcolare una soluzione ottima per un'asta di lunghezza $\ell = 2$ come segue:

Iterazione	j	ℓ	$s[j][\ell]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	Azione
1	4	2	0	0	0	0	0	$j \leftarrow 3$
2	3	2	0	0	0	0	0	$j \leftarrow 2$
3	2	2	1	0	1	0	0	$x[2] \leftarrow 1$, $\ell \leftarrow 2 - 2 = 0$

Una soluzione ottima è il vettore:

$$(0, 1, 0, 0)$$

Numero di passi

- Sia $S(n)$ il numero di passi eseguiti da `DP2` nel caso peggiore, dove n è la lunghezza della asta.
- Calcoliamo $S(n)$ contando quante volte vengono eseguite le righe dello pseudocodice di `DP2`.

Algorithm: DP2 (1/2)

Input : Un'istanza dell'RCP data dalla lunghezza n dell'asta e da un array $p[1..n]$ dei profitti dei pezzi

Output : Il valore ottimo dell'istanza dell'RCP

```
1 function DP2(n, p)
2 |   v := new array[0..n][0..n]
3 |   v[0][0] ← 0
4 |   for ℓ := 1 to n do
5 |   |   v[0][ℓ] ← −∞
6 |   end
```

Algorithm: DP2 (2/2)

```
function DP2(n, p) (continua)
7     for j := 1 to n do
8         for ℓ := 0 to n do
9             if j > ℓ then
10                v[j][ℓ] ← v[j − 1][ℓ]
11            else
12                if p[j] + v[j][ℓ − j] > v[j − 1][ℓ] then
13                    v[j][ℓ] ← p[j] + v[j][ℓ − j]
14                else
15                    v[j][ℓ] ← v[j − 1][ℓ]
16                end
17            end
18        end
19    end
20    return v[n][n]
21 end
```

Numero di esecuzioni delle righe (1/2)

Il numero di volte in cui viene eseguita ogni riga di codice è:

- Riga 2 viene eseguita 1 volta.
- Riga 3 viene eseguita 1 volta.
- Riga 4 viene eseguita n volte.
- Riga 5 viene eseguita n volte.
- Riga 7 viene eseguita n volte.
- Riga 8 viene eseguita $n(n+1)$ volte.
- Riga 9 viene eseguita $n(n+1)$ volte.

Numero di esecuzioni delle righe (2/2)

- Riga 12 viene controllata solo quando $j \leq \ell$, cioè

$$\sum_{j=1}^{n}(n-j+1) = \sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

  volte.

- Per ogni coppia (j, ℓ) viene eseguita una fra la Riga 10, la Riga 13 e la Riga 15, e quindi queste righe di assegnamento vengono eseguite $n(n+1)$ volte in totale.

- Riga 20 viene eseguita 1 volta.

Il numero totale di passi

Quindi il numero totale di passi nel caso peggiore è

$$1 + 1 + n + n + n + n(n+1) + n(n+1) + \frac{n(n+1)}{2} + n(n+1) + 1.$$

Quindi

$$S(n) = 3 + 3n + 3n(n+1) + \frac{n(n+1)}{2} = \frac{7n^2 + 13n + 6}{2}.$$

Quindi

$$S(n) = \Theta(n^2).$$

Il numero di passi è quadratico

Infatti,

$$\lim_{n\to\infty} \frac{\frac{7n^2+13n+6}{2}}{n^2} = \lim_{n\to\infty} \frac{1}{2}\left(7 + \frac{13}{n} + \frac{6}{n^2}\right) = \frac{7}{2} \in \mathbb{R}_{>0}.$$

L'algoritmo di PD esegue

$$S(n) = \Theta(n^2)$$

passi nel caso peggiore.

Riepilogo (1/2)

Passo 1: il valore ottimo:

- $v(j, \ell) = \max\{v(j-1, \ell),\ p_j + v(j, \ell - j)\}$ se $1 \leq j \leq \ell$, e $v(j, \ell) = v(j-1, \ell)$ se $1 \leq \ell < j$
- casi base $v(j, 0) = 0$ e $v(0, \ell) = -\infty$ per $\ell > 0$
- al più 2 sottoproblemi per chiamata; l'albero di ricorsione top-down per $v(4, 4)$ ha 25 chiamate
- bottom-up, riga per riga: sull'istanza di riferimento $v[4][4] = 10$

Riepilogo (2/2)

Passo 2: una soluzione ottima:

- $s[j][\ell] = 1$ se si usa un pezzo di lunghezza j, $s[j][\ell] = 0$ altrimenti
- da (n, n): $j \leftarrow j - 1$ se $s[j][\ell] = 0$; $x_j \leftarrow x_j + 1$, $\ell \leftarrow \ell - j$ se $s[j][\ell] = 1$, finché $\ell = 0$
- sull'istanza di riferimento: $(0, 2, 0, 0)$ per $\ell = 4$, $(0, 0, 1, 0)$ per $\ell = 3$, $(0, 1, 0, 0)$ per $\ell = 2$

Numero di passi:

- `DP2` esegue $S(n) = \dfrac{7n^2 + 13n + 6}{2} = \Theta(n^2)$ passi nel caso peggiore

Errori tipici

Introduzione: Errori tipici

- Raccogliamo gli errori che si fanno più spesso con l'RCP e con i suoi algoritmi di PD
- Uno riguarda il conteggio delle soluzioni, uno il caso base, uno la seconda ricorrenza e uno la ricostruzione di una soluzione
- Controllateli prima di eseguire a mano gli algoritmi di PD

Errori tipici (1/2)

Osservazione 1: Contare i vettori invece degli schemi di taglio

I 2^{n-1} schemi di taglio di un'asta di lunghezza n sono più dei vettori soluzione quando $n \geq 3$, perché schemi diversi possono dare lo stesso vettore.

Osservazione 2: Dimenticare il caso base

$v(0) = 0$ si usa quando il primo pezzo ha la lunghezza dell'intera asta: $v(\ell)$ contiene il termine $p_\ell + v(0)$.

Errori tipici (2/2)

Osservazione 3: Usare la riga $j-1$ nella seconda ricorrenza

Il termine $p_j + v(j, \ell - j)$ resta nella riga j: altrimenti ogni lunghezza dei pezzi si potrebbe tagliare una sola volta.

Osservazione 4: Fermare la ricostruzione troppo presto

La regola $x_{s[\ell]} \leftarrow x_{s[\ell]} + 1$, $\ell \leftarrow \ell - s[\ell]$ si ripete finché $\ell = 0$; l'ultimo pezzo completa l'asta.

Riepilogo

Soluzioni e caso base:

- schemi di taglio diversi possono dare lo stesso vettore: si contano separatamente
- $v(0) = 0$ serve per il termine $p_\ell + v(0)$ dell'intera asta

Seconda ricorrenza e ricostruzione:

- il termine $p_j + v(j, \ell - j)$ resta nella riga j: una lunghezza dei pezzi si può tagliare più volte
- la ricostruzione prosegue finché la lunghezza residua è 0

In sintesi

Introduzione: In sintesi

- Raccogliamo le idee chiave del capitolo sugli algoritmi di PD per l'RCP
- Il quadro riassuntivo raccoglie il risultato sull'istanza di riferimento del capitolo

Riepilogo (1/2)

- L'RCP chiede come tagliare un'asta di lunghezza n in pezzi, con ogni lunghezza disponibile un numero qualsiasi di volte, per massimizzare il profitto totale.
- La ricorrenza $v(\ell) = \max_{j \leq \ell}\{p_j + v(\ell - j)\}$ ha sottostruttura ottima e sottoproblemi sovrapposti, e $v(0) = 0$.
- Il vettore v si riempie bottom-up in $\Theta(n^2)$ passi, un numero polinomiale perché l'input contiene n profitti.
- Il vettore s dei primi pezzi dà una soluzione ottima per ogni lunghezza dell'asta senza costi aggiuntivi.

Riepilogo (2/2)

Quadro riassuntivo dell'esempio dell'RCP (l'istanza di riferimento, asta di lunghezza 4):

Metodo	Lower bound	Upper bound	Origine
Algoritmo di PD	10	10	valore ottimo, due pezzi di lunghezza 2