DP for the rod cutting problem

Outline

Instances and feasible solutions
- Encoding of feasible solutions

A first DP algorithm
- Number of steps of the first DP algorithm

A second DP algorithm
- Number of steps of the second DP algorithm

Typical mistakes

In summary

Learning objectives

At the end of this chapter you will be able to:

- state the RCP, encode a solution as a vector of numbers of pieces, and explain why cutting patterns and vectors are different objects;
- write the recurrence $v(\ell)$ and explain it, base case included;
- run the DP algorithm by hand with the top-down and the bottom-up approach, and with the running best value m;
- reconstruct an optimal solution from the vector s, for the rod and for every shorter rod;
- write the second recurrence $v(j, \ell)$ on pairs (largest piece length, rod length) and count the steps of the DP algorithm.

Prerequisites

- From this module, the chapter «Principles of dynamic programming»: optimal substructure, overlapping subproblems, recurrence relations, base cases, and the top-down and bottom-up approaches.
- From this module, the chapter «DP for the 0/1 knapsack problem»: the pairs (item, capacity) and the reconstruction of a solution, which return here.
- From the module *Algorithms*: how to read pseudocode and how to count the steps of an algorithm («Running time»).

The problem of this chapter

The problem of this chapter is the
rod cutting problem (RCP):

its definition is on its page.

- We describe the instances and the feasible solutions of the RCP
- We write two DP algorithms: the first one on the rod lengths ℓ, the second one on the pairs (j, ℓ) (largest piece length, rod length)

Instances and feasible solutions

Introduction: Instances and feasible solutions

- We introduce the RCP instance that we use throughout the chapter
- We count the feasible solutions of an RCP instance, and see why enumerating them quickly becomes impractical
- We encode a feasible solution as a vector of numbers of pieces, and see why different cutting patterns can have the same vector
- This is the starting point of the two DP algorithms of the next sections

Example: RCP instance

We consider a rod of length $n = 4$ and a set $\{1, 2, 3, 4\}$ of 4 possible piece lengths with the following profits:

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

We refer to this RCP instance as the reference instance: we use it in all the examples of the chapter.

How many feasible solutions?

The number of possible solutions of the RCP is 2^{n-1}

since we can cut a rod of length n in 2^{n-1} different ways,

with an independent option of cutting (or not)
at each possible cutting position.

- The number of possible RCP solutions grows exponentially with the length n of the rod:

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

- Therefore, enumerating all possible solutions quickly becomes impractical as n increases.

Feasible solutions

A *feasible solution* of the RCP is a way of cutting the rod into pieces, such that the total length of the pieces is equal to the rod length.

- An RCP feasible solution can be described by an integer solution vector

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

where each component has the following meaning:

$$x_\ell = \text{number of pieces of length } \ell, \qquad \forall \ell \in \{1, 2, \ldots, n\}.$$

Encoding of feasible solutions

- The number of pieces is

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

- Since the solution is feasible, the total length of the pieces must be equal to the rod length:

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

- The total profit of a feasible solution is

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

Cutting patterns and vectors

- Two or more cutting patterns may have the same vector representation: the vector records how many pieces of each length are obtained, but it does not specify the order in which the pieces are cut.
- Cutting patterns with the same vector representation have the same total profit, since the order in which the pieces are cut does not affect the obtained pieces and therefore does not affect the total profit.

Next: all the feasible solutions of the reference instance.

Example: enumeration of all RCP solutions (1/5)

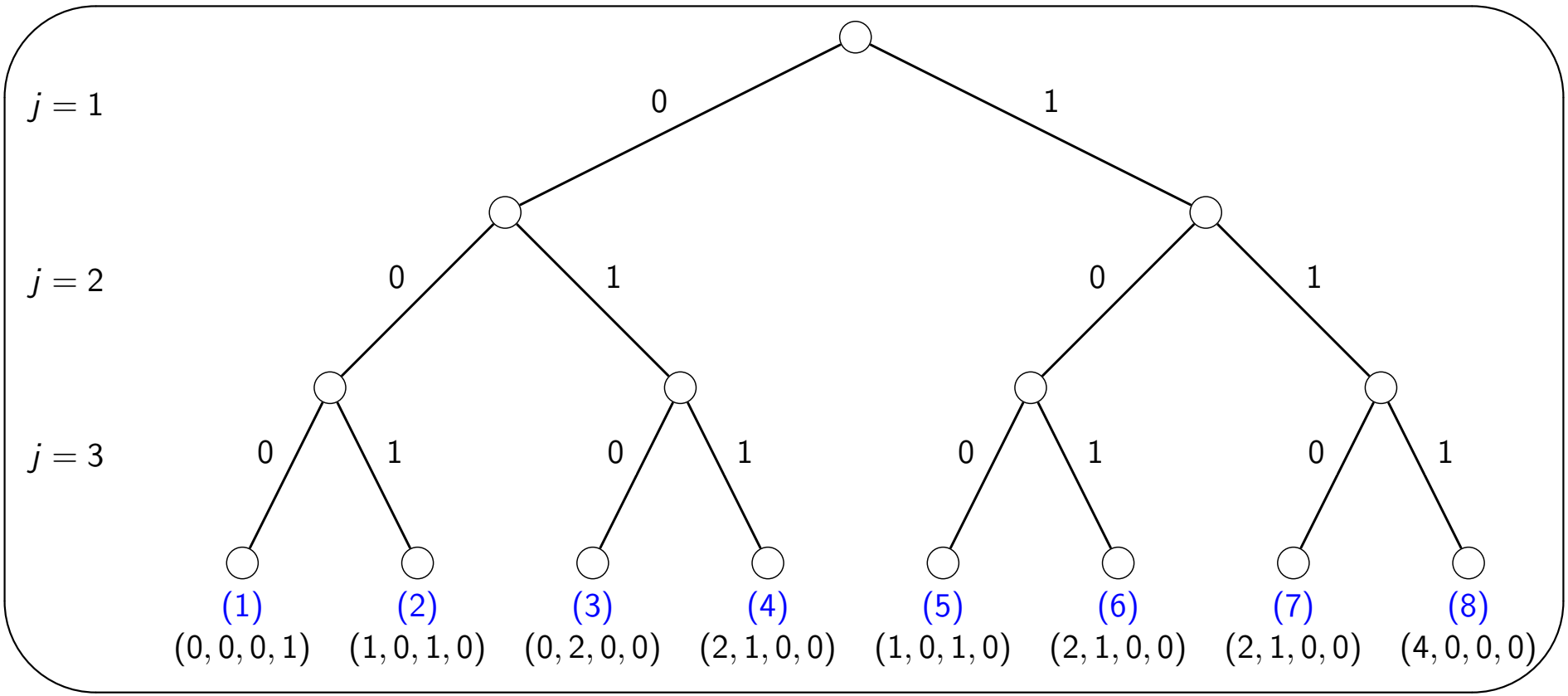
We consider the reference instance, with a rod of length $n = 4$ and the following profits of the possible piece lengths:

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

- There are $2^{4-1} = 8$ feasible solutions, corresponding to the different ways of cutting the rod.
- In the binary tree of the next slide, each level corresponds to a possible cutting position: an edge labeled 1 means that the rod is cut at that length, while an edge labeled 0 means that it is not cut at that length.
- Each leaf represents one complete cutting pattern and reports the corresponding vector.

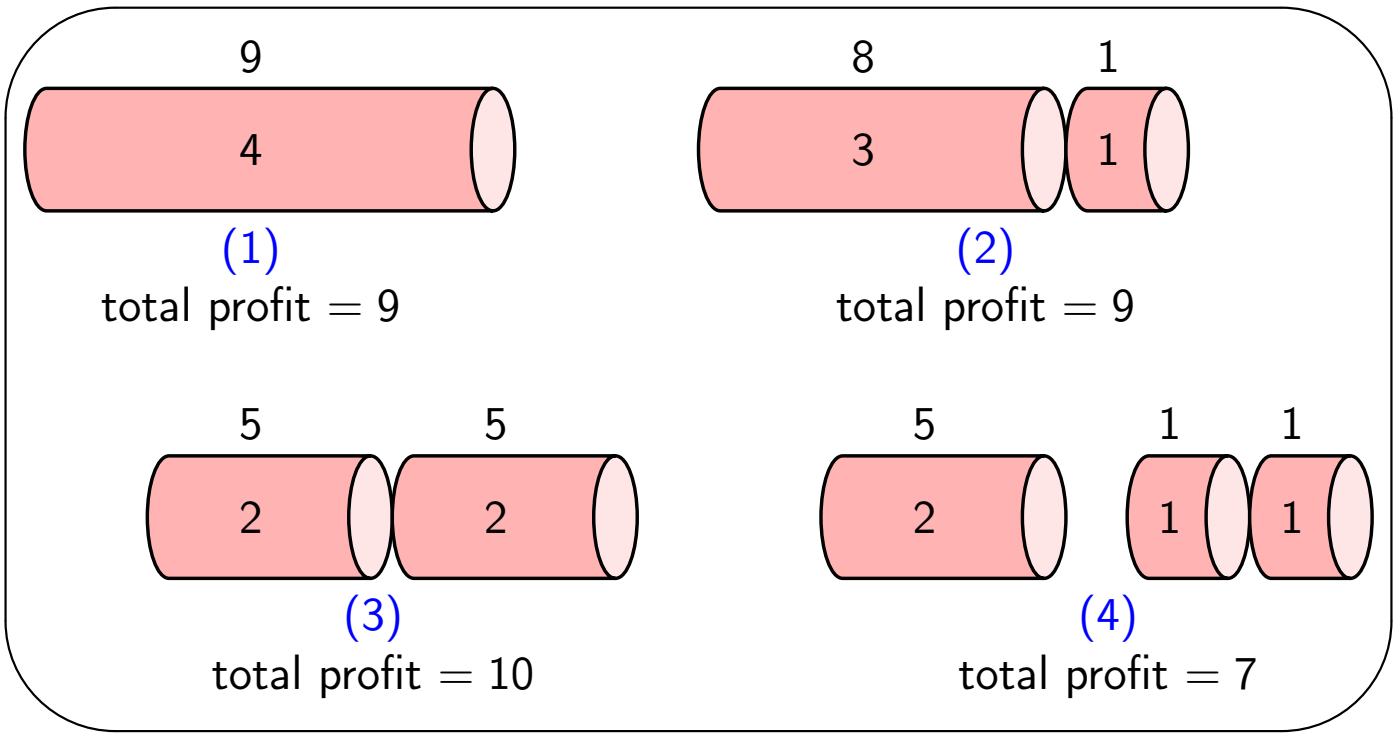
Example: enumeration of all RCP solutions (2/5)

The binary tree of the cutting decisions at the positions $j = 1, 2, 3$:



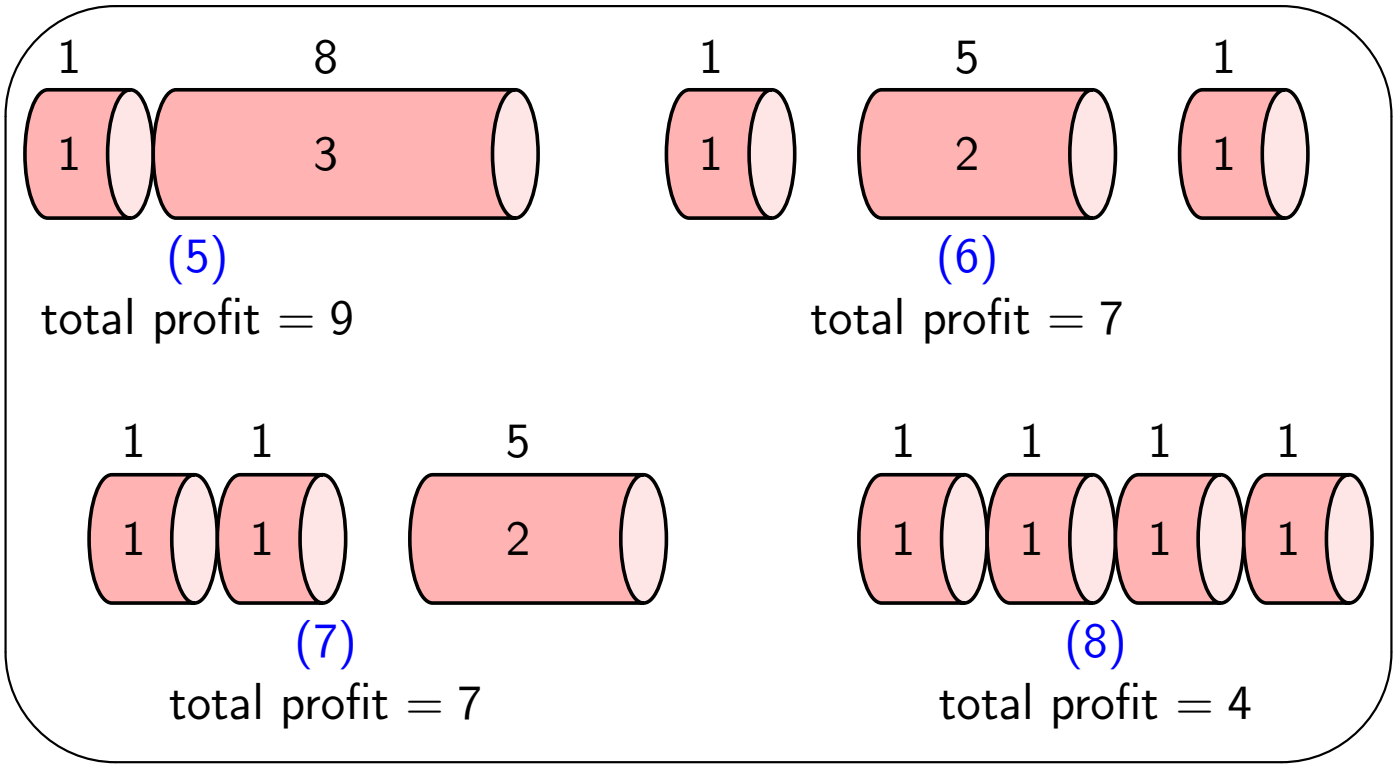
Example: enumeration of all RCP solutions (3/5)

The next figures show all possible cutting patterns associated with the leaves of the tree, through the labels $(1), (2), \ldots, (8)$. For each obtained piece, the number written inside the piece is its length, while the number above it is its profit.



Example: enumeration of all RCP solutions (4/5)

The cutting patterns $(5), (6), (7), (8)$:



Example: enumeration of all RCP solutions (5/5)

Label	Solution vector	Number of pieces	Total profit
(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	$1+1+1+1=4$

Summary

Instances and feasible solutions:

- An RCP instance: a rod of length n and a profit p_ℓ for each piece length $\ell \in \{1, 2, \ldots, n\}$
- A rod of length n can be cut in $2^{n-1} = O(2^n)$ ways: too many to enumerate

Encoding:

- An integer vector $(x_1, x_2, \ldots, x_n)$, with x_ℓ pieces of length ℓ, $\sum_{\ell=1}^{n} \ell\, x_\ell = n$ and profit $\sum_{\ell=1}^{n} p_\ell x_\ell$
- Different cutting patterns can have the same vector, and then the same total profit

A first DP algorithm

Introduction: A first DP algorithm

- We write a recurrence $v(\ell)$ over the rod lengths, and verify optimal substructure and overlapping subproblems
- Step 1: we compute the optimal value, top-down and bottom-up
- Step 2: we store the first piece of each rod length and reconstruct an optimal solution
- Finally, we count the steps of the DP algorithm

Step 1: computation of the optimal value

For any $\ell \in \{0, 1, \ldots, n\}$, let $v(\ell)$ be the optimal value of the RCP subinstance given by:

1. a rod of length ℓ,
2. a set $\{1, 2, \ldots, \ell\}$ of ℓ possible piece lengths,
3. for each piece length $j \in \{1, 2, \ldots, \ell\}$, a profit p_j.

The recursive formula for $v(\ell)$

The recursive formula to compute $v(\ell)$ reads as follows:

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

Independent subproblems and base case

- The recursive decomposition process of the recursive formula for $v(\ell)$ generates ℓ **independent subproblems**, each corresponding to a possible choice of cutting a first piece of length j from a rod of length ℓ.
- Here, $v(0) = 0$ since a rod of length 0 cannot be cut and has profit 0.

Optimal substructure

- The optimal value $v(\ell)$ is obtained by trying all possible first-piece lengths and selecting the one that maximizes the sum $p_j + v(\ell - j)$ of the profit of the first piece and the optimal value of the residual subproblem.

- Hence, an optimal solution for a rod of length ℓ is built by combining the chosen first piece of length j with an optimal solution of the residual subproblem of length $\ell - j$.

$\rightsquigarrow$ **Optimal Substructure verified!**

Overlapping subproblems

- Different choices of the first-piece length, possibly made for different values of ℓ, can lead to the same residual length.
- Therefore the same value $v(\ell)$ may need to be computed several times during the recursive evaluation of the recursive formula for $v(\ell)$.

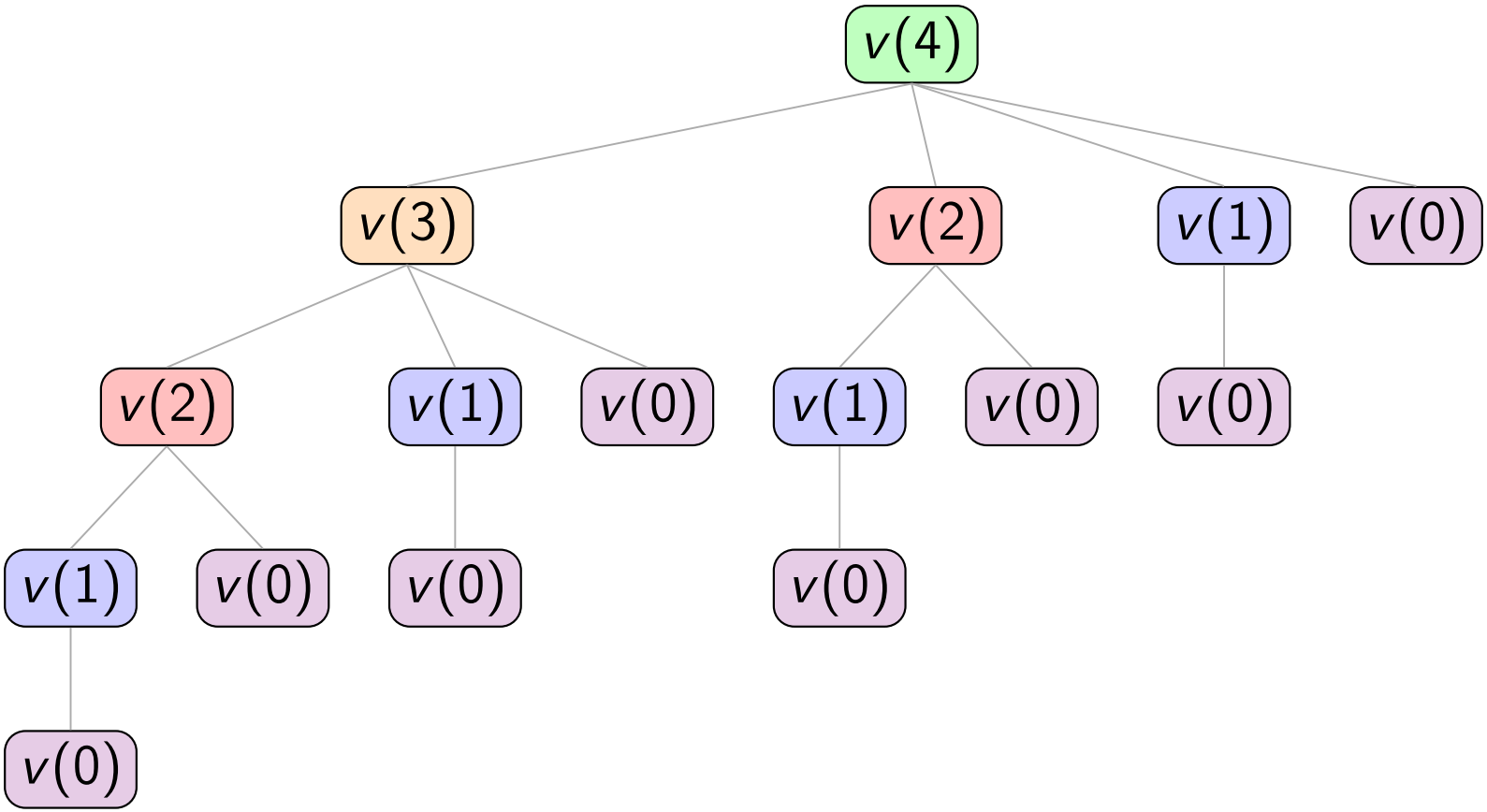
$\rightsquigarrow$ **Overlapping Subproblems verified!**

The top-down approach

- A **first method** to compute the optimal value $v(n)$ of the RCP instance is the **top-down approach**.
- The method starts from the call with $\ell = n$ and evaluates the recursive formula for $v(\ell)$ recursively: each call for a rod of length ℓ triggers further calls for rods of length $\ell - j$, for all $j \in \{1, 2, \ldots, \ell\}$, until the base case $v(0)$ is reached.
- The **recursion tree** of the function v has one node for each recursive call, labeled with the function call it represents. The root represents the initial call. The children of a node represent the recursive calls made by that call. The leaf nodes represent the base cases.

Example: recursion tree (1/2)

We consider the reference instance. This is the recursion tree of the top-down approach to compute $v(4)$:



Example: recursion tree (2/2)

The function v is called 16 times, and the same subproblems are solved multiple times:

- $v(2)$ is called 2 times,
- $v(1)$ is called 4 times,
- $v(0)$ is called 8 times.

All the other calls are made only once.

Computing the values in increasing order of length

The value $v(\ell)$, for a generic rod length ℓ, can be computed once the values $v(j)$ are known for all smaller lengths $j < \ell$.

Example: computing the values of the v function (1/2)

We consider the reference instance with a rod of length $n = 4$ and the following profits of the possible piece lengths:

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

The value of the function v can be computed by starting from the base case $v(0) = 0$ and then computing the values of $v(1)$, $v(2)$, $v(3)$, and $v(4)$ in order, as follows:

- Since we know the value of $v(0)$, we can compute $v(1)$, as follows:

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

- Once we have computed $v(1)$, we can compute $v(2)$, as follows:

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

Example: computing the values of the v function (2/2)

- Once we have computed $v(2)$ and $v(1)$, we can compute $v(3)$, as follows:

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

- Once we have computed $v(3)$, $v(2)$ and $v(1)$, we can compute $v(4)$, as follows:

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

The lookup table

- We can avoid solving the same subproblems repeatedly by storing the value of each subproblem as soon as it is computed. The values are stored in a **lookup table**.
- Since solving a subproblem with value $v(\ell)$ depends only on subproblems with values $v(j)$, with $j < \ell$, we can solve the subproblems for increasing values of ℓ, starting from the base case $v(0) = 0$ and building up to $v(n)$.
- This observation naturally leads to a **much more efficient method** to compute the optimal value $v(n)$ of the RCP instance.

The bottom-up approach

- A **second method** to compute the optimal value $v(n)$ of the RCP instance is the **bottom-up approach**.
- The method starts from the base case $v(0) = 0$ and evaluates the recursive formula for $v(\ell)$ for increasing rod lengths $\ell \in \{1, 2, \ldots, n\}$: each value $v(\ell)$ is computed using the already-computed values $v(0), v(1), \ldots, v(\ell - 1)$, until the value $v(n)$ of the RCP instance is obtained.

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**.

Algorithm: DP1

Input : An RCP instance given by the length n of the rod and an array $p[1..n]$ of piece profits

Output : The optimal value of the RCP instance

```
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
```

How the DP algorithm works

- The vector $v[0..n]$ is the **lookup table**: it stores the optimal values of all subproblems, with the base case $v[0] = 0$.
- The outer loop, over $\ell \in \{1, 2, \ldots, n\}$, computes $v[\ell]$ in increasing order of rod length.
- The inner loop, over $j \in \{1, 2, \ldots, \ell\}$, evaluates all possible first-piece lengths and keeps in m the largest value $p[j] + v[\ell - j]$.
- At the end of the inner loop, $v[\ell] \leftarrow m$ stores the optimal value $v[\ell]$.
- Finally, the optimal value $v[n]$ of the RCP instance is returned.

Example: execution of the DP algorithm (1/4)

We consider the reference instance with a rod of length $n = 4$ and the following profits of the possible piece lengths:

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

For each step of the DP algorithm, the next tables show:

- the values of ℓ and j, the cell $v[\ell]$, and the current value of m;
- the cell of m is highlighted in red when updated;
- the cell of $v[\ell]$ is highlighted in blue when assigned.

Example: execution of the DP algorithm (2/4)

Initialization and rod lengths $\ell = 1$ and $\ell = 2$:

ℓ	j	$v[\ell]$	m	Action
–	–	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	$v[2] \leftarrow m = 5$

Example: execution of the DP algorithm (3/4)

Rod lengths $\ell = 3$ and $\ell = 4$:

ℓ	j	$v[\ell]$	m	Action
	–	–	0	$m \leftarrow 0$
3	1	–	6	$p[1]+v[2] = 1+5 = 6 > 0$: $m \leftarrow 6$
	2	–	6	$p[2]+v[1] = 5+1 = 6 = m$: no update
	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	$p[3]+v[1] = 8+1 = 9 < 10$: no update
	4	–	10	$p[4]+v[0] = 9+0 = 9 < 10$: no update
	–	10	10	$v[4] \leftarrow m = 10$

Example: execution of the DP algorithm (4/4)

This is the vector v after the execution of the DP algorithm:

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

- After the execution of the DP algorithm, the vector v contains the optimal value $v[\ell]$ for every rod length.
- The optimal value of the reference instance is $v[4] = 10$.

Step 2: computation of an optimal solution

- To compute an optimal solution, it is necessary to store some additional information.
- For each rod length ℓ, in addition to the optimal value $v[\ell]$, we store $s[\ell]$, the size of the first piece to cut off in an optimal solution for a rod of length ℓ.
- Starting from $\ell = n$, the value $s[\ell]$ gives the length of the first piece to cut; the remaining rod has length $\ell - s[\ell]$, and the same rule can be applied again, and so on, until the residual length reaches 0.

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**
with the storing of the information necessary to compute an optimal solution.

Algorithm: DP1_InfoSol (1/2)

Input : An RCP instance given by the length n of the rod and an array $p[1..n]$ of piece profits

Output: The optimal value of the RCP instance and the information vector $s[1..n]$

```
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) (continued)

```
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
```

How the information is stored

- The DP algorithm uses the additional temporary variable t to keep track of the decision made when m is updated.
- If a first piece of length j is selected, $t \leftarrow j$ stores this decision.
- At the end of the inner loop, $s[\ell] \leftarrow t$ stores the length of the selected first piece, together with the optimal value $v[\ell] \leftarrow m$.

Example: computation of the additional information (1/4)

We consider the reference instance with a rod of length $n = 4$ and the following profits of the possible piece lengths:

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

For each step of the DP algorithm, the next tables show:

- the values of ℓ and j, the cell $s[\ell]$, the current value of m, and the current value of t;
- the cell of m is highlighted in red when updated;
- the cell of t and the cell of $s[\ell]$ are highlighted in blue when updated.

Example: computation of the additional information (2/4)

Initial state and rod lengths $\ell = 1$ and $\ell = 2$:

ℓ	j	$s[\ell]$	m	t	Action
–	–	–	–	–	Initial state
	–	–	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	$v[2] \leftarrow m = 5,\ s[2] \leftarrow t = 2$

Example: computation of the additional information (3/4)

Rod lengths $\ell = 3$ and $\ell = 4$:

ℓ	j	$s[\ell]$	m	t	Action
	–	–	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	$5{+}1 = 6 = m$: no update
	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	$8{+}1 = 9 < 10$: no update
	4	–	10	2	$9{+}0 = 9 < 10$: no update
	–	2	10	2	$v[4] \leftarrow m = 10,\ s[4] \leftarrow t = 2$

Example: computation of the additional information (4/4)

This is the vector s after the execution of the DP algorithm:

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

- For example, $s[4] = 2$: in an optimal solution for the rod of length 4, the first piece to cut off has length 2.

Computing an optimal solution from the vector s

- Once the vector s has been computed, an optimal solution of the RCP instance can be computed by repeatedly applying the following rule, starting from $j = n$ and $x_\ell = 0$ for all $\ell \in \{1, 2, \ldots, n\}$:

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

  until $j = 0$.

- This yields the vector $(x_1, x_2, \ldots, x_n)$, where each entry x_ℓ is the number of pieces of length ℓ in an optimal solution.

- The following pseudo-code implements the computation of an optimal solution, given the vector s.

Algorithm: DP1_SolComputation

Input : The rod length n of the RCP instance and the vector $s[1..n]$

Output: An optimal solution of the RCP instance

```
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
```


Sapienza Università di Roma

Example: computing an optimal solution

After the execution of the DP algorithm on the reference instance, we have the following vector s:

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

We can compute an optimal solution for a rod of length $n = 4$ as follows:

Iteration	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	$j = 2 - 2 = 0$

An optimal solution is the vector:

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

Optimal solutions for every rod length

The same computation procedure, based on the vector s, can be applied starting from any rod length ℓ.

- The vector s is computed once, by the DP algorithm.
- It gives an optimal solution for every rod length $\ell \in \{1, 2, \ldots, n\}$.

Example: optimal solutions for lengths 3 and 2 (1/2)

After the execution of the DP algorithm on the reference instance, we have the following vector s:

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

We can compute an optimal solution for a rod of length 3 as follows:

Iteration	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	$j = 3 - 3 = 0$

An optimal solution is the vector:

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

Example: optimal solutions for lengths 3 and 2 (2/2)

We can compute an optimal solution for a rod of length 2 as follows:

Iteration	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	$j = 2 - 2 = 0$

An optimal solution is the vector:

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

Number of steps

- Let $S(n)$ denote the number of steps executed by `DP1` in the worst case, where n is the rod length.
- We compute $S(n)$ by counting how many times Line 2–Line 13 are executed.
- In the worst case, the condition at line 7 is true at every iteration of the inner loop, and therefore line 8 is executed as often as line 7.

Algorithm: DP1

Input : An RCP instance given by the length n of the rod and an array $p[1..n]$ of piece profits

Output : The optimal value of the RCP instance

```
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
```

Number of executions of the lines

The number of executions of the relevant lines is:

- Line 2 is executed 1 time.
- Line 3 is executed 1 time.
- Line 4 is executed n times.
- Line 5 is executed n times.
- Line 6 is executed $\sum_{\ell=1}^{n} \ell$ times.
- Line 7 is executed $\sum_{\ell=1}^{n} \ell$ times.
- Line 8 is executed $\sum_{\ell=1}^{n} \ell$ times.
- Line 11 is executed n times.
- Line 13 is executed 1 time.

The total number of steps

Therefore, the total number of steps in the worst case is

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

Using $\sum_{\ell=1}^{n} \ell = \dfrac{n(n+1)}{2}$, we obtain

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

Hence,

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

The number of steps is quadratic

Indeed,

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

The DP algorithm executes

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

steps in the worst case.

Summary (1/2)

Step 1: the optimal value:

- $v(\ell) = \max\{p_j + v(\ell - j) : j \in \{1, 2, \ldots, \ell\}\}$ for $1 \leq \ell \leq n$, with the base case $v(0) = 0$
- optimal substructure and overlapping subproblems: the top-down recursion tree for $v(4)$ has 16 calls
- bottom-up: the vector v is filled for increasing ℓ; on the reference instance $v[4] = 10$

Step 2: an optimal solution:

- $s[\ell]$ is the length of the first piece to cut off from a rod of length ℓ
- rule $x_{s[j]} \leftarrow x_{s[j]} + 1$, $j \leftarrow j - s[j]$ until $j = 0$

Summary (2/2)

Solutions on the reference instance:

- $s = (1, 2, 3, 2)$
- rod of length 4: $(0, 2, 0, 0)$, two pieces of length 2, profit 10
- rods of length 3 and 2: $(0, 0, 1, 0)$ and $(0, 1, 0, 0)$

Number of steps:

- `DP1` executes $S(n) = \dfrac{3n^2 + 9n + 6}{2} = \Theta(n^2)$ steps in the worst case

A second DP algorithm

Introduction: A second DP algorithm

- We write a second recurrence $v(j, \ell)$, on the pairs (largest piece length, rod length), as in the DP algorithm of the 0/1 knapsack problem
- For each piece length j we decide whether to cut a piece of length j or not: at most 2 subproblems instead of ℓ
- We follow the same steps: optimal value top-down and bottom-up, optimal solution, number of steps

Step 1: computation of the optimal value

For any $j \in \{0, 1, \ldots, n\}$ and $\ell \in \{0, 1, \ldots, n\}$, let $v(j, \ell)$ be the optimal value of the RCP subinstance given by:

1. a rod of length ℓ,
2. a set $\{1, 2, \ldots, j\}$ of j possible piece lengths,
3. for each piece length $k \in \{1, 2, \ldots, j\}$, a profit p_k.

The recursive formula for $v(j, \ell)$

The recursive formula to compute $v(j, \ell)$ reads as follows:

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

Independent subproblems and base cases

- The recursive decomposition process of the recursive formula for $v(j, \ell)$ generates at most 2 **independent subproblems**, corresponding to whether or not to cut a piece of length j from a rod of length ℓ. If $j > \ell$, a piece of length j cannot be cut from the rod, and therefore the only feasible option is not to cut it.

- Here, $v(j, 0) = 0$ since a rod of length 0 cannot be cut and has profit 0.

- Moreover, $v(0, \ell) = -\infty$ for $\ell > 0$: in this case no positive piece length is available, so no feasible solution exists for a rod of positive length. The value $-\infty$ is used to rule out this infeasible case.

Optimal substructure

- If $1 \leq j \leq \ell$, the optimal value $v(j, \ell)$ is obtained by trying the two possible options, cutting or not cutting a first piece of length j, and selecting the one that maximizes the profit. The corresponding values are $p_j + v(j, \ell - j)$ and $v(j-1, \ell)$, respectively.
- If $1 \leq \ell < j$, piece length j cannot be used, and therefore $v(j, \ell) = v(j-1, \ell)$.
- Hence, an optimal solution for a rod of length ℓ with largest available piece length j is built by combining the chosen decision on piece length j with an optimal solution of the corresponding residual subproblem.

$\rightsquigarrow$ **Optimal Substructure verified!**

Overlapping subproblems

- Different sequences of decisions on the piece lengths, possibly made for different values of j and ℓ, can lead to the same subproblem.
- Therefore the same value $v(j, \ell)$ may need to be computed several times during the recursive evaluation of the recursive formula for $v(j, \ell)$.

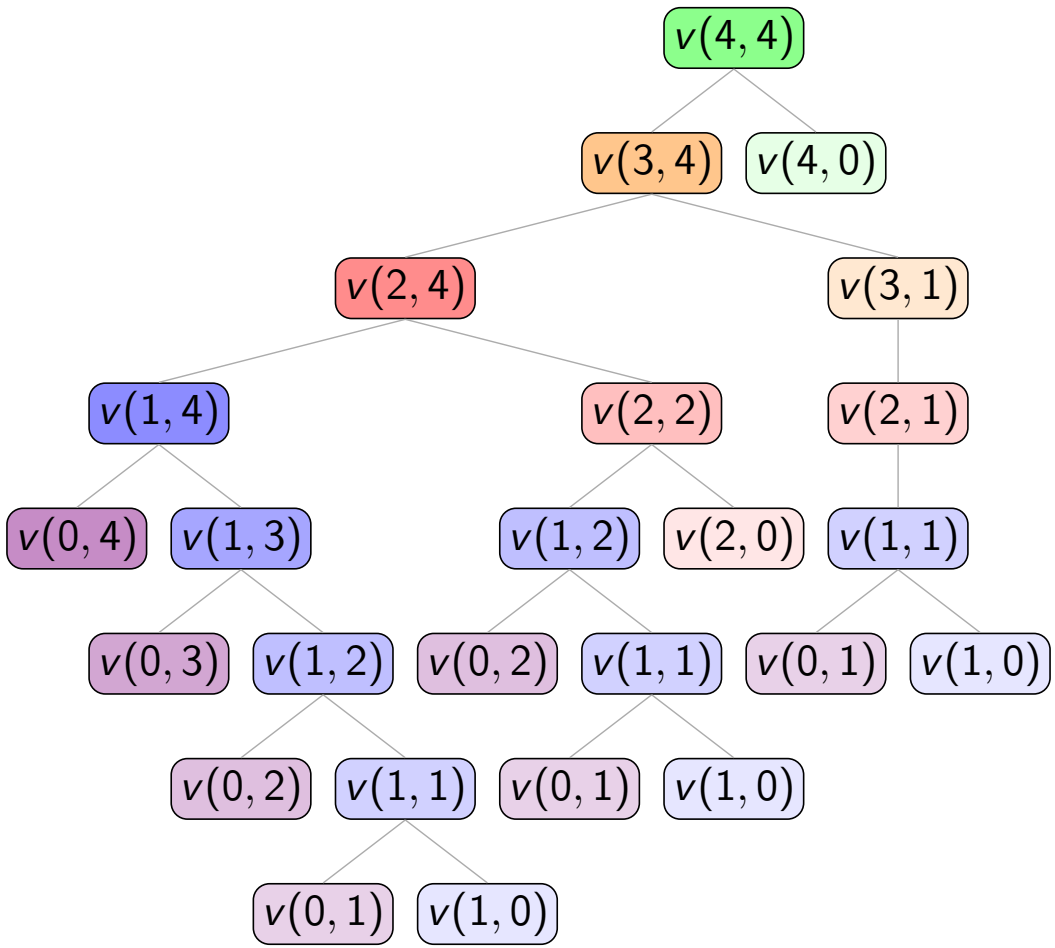
$\rightsquigarrow$ **Overlapping Subproblems verified!**

The top-down approach

- A **first method** to compute the optimal value $v(n, n)$ of the RCP instance is the **top-down approach**.
- The method starts from the call with $(j, \ell) = (n, n)$ and evaluates the recursive formula for $v(j, \ell)$ recursively: each call for a rod of length ℓ and largest available piece length j triggers further calls corresponding to the feasible decisions on piece length j, until one of the base cases $v(j, 0)$ or $v(0, \ell)$ is reached.
- The **recursion tree** of the function v has one node for each recursive call, labeled with the function call it represents. The root represents the initial call. The children of a node represent the recursive calls made by that call. The leaf nodes represent the base cases.

Example: recursion tree (1/2)

We consider the reference instance. This is the recursion tree of the top-down approach to compute $v(4, 4)$:



Example: recursion tree (2/2)

The function v is called 25 times, and the same subproblems are solved multiple times:

- $v(1, 2)$ is called 2 times,
- $v(0, 2)$ is called 2 times,
- $v(1, 1)$ is called 3 times,
- $v(0, 1)$ is called 3 times,
- $v(1, 0)$ is called 3 times.

All the other calls are made only once.

Computing the values row by row

- The value $v(j, \ell)$ can be computed once $v(j-1, \ell)$ is known and, when $j \leq \ell$, $v(j, \ell - j)$ is also known.
- Therefore, for a fixed value of j, all values $v(j, \ell)$ can be computed by considering the values of ℓ in increasing order.
- Starting from the base cases $v(j, 0) = 0$ and $v(0, \ell) = -\infty$, the values of the function v can be computed row by row, moving to the next value of j only after all values $v(j, \ell)$ for the current value of j have been computed.

Example: computing the values of the v function (1/4)

We consider the reference instance, with the profits $p_1 = 1$, $p_2 = 5$, $p_3 = 8$, $p_4 = 9$. The values of the function v are computed as follows.

For $j = 1$, only pieces of length 1 can be used. Therefore:

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

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

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

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

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

Example: computing the values of the v function (2/4)

For $j = 2$, pieces of length 1 and 2 can be used. We obtain:

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

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

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

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

Example: computing the values of the v function (3/4)

For $j = 3$, pieces of length 1, 2, and 3 can be used. We obtain:

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

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

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

Example: computing the values of the v function (4/4)

For $j = 4$, all piece lengths are available. We obtain:

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

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

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

The lookup table

- We can avoid solving the same subproblems repeatedly by storing the value of each subproblem as soon as it is computed. The values are stored in a **lookup table**.

- Since solving any particular subproblem depends only on subproblems with a smaller value of j, or with the same value of j and a smaller rod length, we can solve the subproblems in increasing order of j and, for each fixed value of j, in increasing order of rod length, starting from the base cases $v(0,0) = 0$ and $v(0,\ell) = -\infty$ for all $\ell \in \{1,\ldots,n\}$ and building up to $v(n,n)$.

- This observation naturally leads to a **much more efficient method** to compute the optimal value $v(n,n)$ of the RCP instance.

The bottom-up approach

- A **second method** to compute the optimal value $v(n, n)$ of the RCP instance is the **bottom-up approach**.
- The method starts from the base cases $v(0, 0) = 0$ and $v(0, \ell) = -\infty$ for all $\ell \in \{1, \ldots, n\}$ and evaluates the recursive formula for $v(j, \ell)$ for increasing values of $j \in \{1, 2, \ldots, n\}$ and $\ell \in \{0, 1, \ldots, n\}$: each value $v(j, \ell)$ is computed using already-computed values, until the value $v(n, n)$ of the RCP instance is obtained.

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**.

Algorithm: DP2 (1/2)

Input : An RCP instance given by the length n of the rod and an array $p[1..n]$ of piece profits

Output : The optimal value of the RCP instance

```
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)* (continued)

```
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
```

How the DP algorithm works (1/2)

- The matrix $v[0..n][0..n]$ is the **lookup table**: it stores the optimal values of all subproblems.
- The initialization sets $v[0][0] = 0$ and $v[0][\ell] = -\infty$ for all rod lengths $\ell \in \{1, \ldots, n\}$, since a positive length cannot be filled if no piece length is available.
- The outer loop, over $j \in \{1, 2, \ldots, n\}$, considers the available piece lengths in increasing order.
- The inner loop, over $\ell \in \{0, 1, \ldots, n\}$, computes the best value for each rod length using only pieces of length at most j.

How the DP algorithm works (2/2)

- If $j > \ell$, piece length j cannot be used, and therefore $v[j][\ell] \leftarrow v[j-1][\ell]$.
- Otherwise, the DP algorithm compares the value obtained by not using piece length j, namely $v[j-1][\ell]$, with the value obtained by using one piece of length j, namely $p[j] + v[j][\ell - j]$.
- Finally, the optimal value of the RCP instance is returned as $v[n][n]$.

Example: execution of the DP algorithm (1/5)

We consider the reference instance with a rod of length $n = 4$ and the following profits of the possible piece lengths:

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

For each step of the DP algorithm, the next tables show the values of j and ℓ, the cell $v[j][\ell]$, and the operation performed.

Example: execution of the DP algorithm (2/5)

Rows $j = 0$ and $j = 1$:

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

Example: execution of the DP algorithm (3/5)

Rows $j = 2$ and $j = 3$:

j	ℓ	$v[j][\ell]$	Action
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	$\max\{v[2][4], p[3] + v[3][1]\} = \max\{10, 8 + 1\} = 10$

Example: execution of the DP algorithm (4/5)

Row $j = 4$:

j	ℓ	$v[j][\ell]$	Action
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	$\max\{v[3][4], p[4] + v[4][0]\} = \max\{10, 9 + 0\} = 10$

Example: execution of the DP algorithm (5/5)

This is the matrix v after the execution of the DP algorithm:

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

- After the execution of the DP algorithm, the matrix v contains the optimal value $v[j][\ell]$ for every subproblem (j, ℓ).
- The optimal value of the reference instance is $v[4][4] = 10$.

Step 2: computation of an optimal solution

- To compute an optimal solution, it is necessary to store some additional information.
- For each value j of the largest available piece length and each rod length ℓ, in addition to the optimal value $v[j][\ell]$, we store $s[j][\ell]$, which is equal to 1 if one piece of length j is used and to 0 otherwise.
- Starting from $j = n$ and $\ell = n$, if $s[j][\ell] = 0$, then piece length j is not used and j is decreased by 1. If $s[j][\ell] = 1$, then one piece of length j is used and the residual rod length becomes $\ell - j$. The procedure stops when the residual length reaches 0.

Algorithm: DP2_InfoSol (1/3)

Input : An RCP instance given by the length n of the rod and an array $p[1..n]$ of piece profits

Output : The optimal value of the RCP instance and the matrix of binary information $s[0..n][0..n]$

```
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) (continued)

```
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) (continued)
    for j := 1 to n (continued) do
        for ℓ := 0 to n (continued) do
15          else
16              if p[j] + v[j][ℓ − j] > v[j − 1][ℓ] then
17                  v[j][ℓ] ← p[j] + v[j][ℓ − j]
18                  s[j][ℓ] ← 1
19              else
20                  v[j][ℓ] ← v[j − 1][ℓ]
21                  s[j][ℓ] ← 0
22              end
23          end
24      end
25  end
26  return v[n][n], s
27 end
```

How the information is stored

- The DP algorithm uses the additional matrix s, which keeps track of the decision made when $v[j][\ell]$ is computed.
- If one piece of length j is selected, $s[j][\ell] \leftarrow 1$ stores this decision.
- Otherwise, $s[j][\ell] \leftarrow 0$ stores that piece length j is not used.

Example: computation of the additional information (1/5)

We consider the reference instance with a rod of length $n = 4$ and the following profits of the possible piece lengths:

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

For each step of the DP algorithm, the next tables show the values of j and ℓ, the cell $s[j][\ell]$, and the operation performed.

Example: computation of the additional information (2/5)

Rows $j = 0$ and $j = 1$:

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

Example: computation of the additional information (3/5)

Rows $j = 2$ and $j = 3$:

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

Example: computation of the additional information (4/5)

Row $j = 4$:

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

Example: computation of the additional information (5/5)

This is the matrix s after the execution of the DP algorithm:

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

Computing an optimal solution from the matrix s

- Once the matrix s has been computed, an optimal solution for a rod of length n can be computed by repeatedly applying the following rule, starting from $(j, \ell) = (n, n)$ and $x_h = 0$ for all $h \in \{1, 2, \ldots, n\}$:

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

- The procedure stops when $\ell = 0$. This yields the vector $(x_1, x_2, \ldots, x_n)$, where each entry x_h is the number of pieces of length h in an optimal solution.

- The following pseudo-code implements the computation of an optimal solution, given the matrix s.

Algorithm: DP2_SolComputation (1/2)

Input : The rod length n and the matrix $s[0..n][0..n]$
Output: An optimal solution of the RCP instance

```
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) (continued)

```
 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
```

Example: computing an optimal solution (1/2)

After the execution of the DP algorithm on the reference instance, we have the following matrix 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

Example: computing an optimal solution (2/2)

We can compute an optimal solution for a rod of length $n = 4$ as follows:

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

An optimal solution is the vector:

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

Optimal solutions for every rod length

The same computation procedure, based on the matrix s,
can be applied to any rod length $\ell \leq n$
by starting from $j = n$.

Example: computing optimal solutions (1/3)

After the execution of the DP algorithm on the reference instance, we have the following matrix 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

Example: computing optimal solutions (2/3)

We can compute an optimal solution for a rod of length $\ell = 3$ as follows:

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

An optimal solution is the vector:

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

Example: computing optimal solutions (3/3)

We can compute an optimal solution for a rod of length $\ell = 2$ as follows:

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

An optimal solution is the vector:

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

Number of steps

- Let $S(n)$ denote the number of steps executed by `DP2` in the worst case, where n is the rod length.
- We compute $S(n)$ by counting how many times the lines of the pseudo-code of `DP2` are executed.

Algorithm: DP2 (1/2)

Input : An RCP instance given by the length n of the rod and an array $p[1..n]$ of piece profits

Output : The optimal value of the RCP instance

```
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)* (continued)

```
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
```

Number of executions of the lines (1/2)

The number of times each line of code is executed is:

- Line 2 is executed 1 time.
- Line 3 is executed 1 time.
- Line 4 is executed n times.
- Line 5 is executed n times.
- Line 7 is executed n times.
- Line 8 is executed $n(n+1)$ times.
- Line 9 is executed $n(n+1)$ times.

Number of executions of the lines (2/2)

- Line 12 is checked only when $j \leq \ell$, that is

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

  times.

- One among Line 10, Line 13, and Line 15 is executed for each pair (j, ℓ), and therefore these assignment lines are executed $n(n+1)$ times in total.

- Line 20 is executed 1 time.

The total number of steps

Therefore, the total number of steps in the worst case is

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

Thus,

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

Hence,

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

The number of steps is quadratic

Indeed,

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

The DP algorithm executes

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

steps in the worst case.

Summary (1/2)

Step 1: the optimal value:

- $v(j, \ell) = \max\{v(j-1, \ell),\ p_j + v(j, \ell - j)\}$ if $1 \leq j \leq \ell$, and $v(j, \ell) = v(j-1, \ell)$ if $1 \leq \ell < j$
- base cases $v(j, 0) = 0$ and $v(0, \ell) = -\infty$ for $\ell > 0$
- at most 2 subproblems per call; the top-down recursion tree for $v(4, 4)$ has 25 calls
- bottom-up, row by row: on the reference instance $v[4][4] = 10$

Summary (2/2)

Step 2: an optimal solution:

- $s[j][\ell] = 1$ if one piece of length j is used, $s[j][\ell] = 0$ otherwise
- from (n, n): $j \leftarrow j - 1$ if $s[j][\ell] = 0$; $x_j \leftarrow x_j + 1$, $\ell \leftarrow \ell - j$ if $s[j][\ell] = 1$, until $\ell = 0$
- on the reference instance: $(0, 2, 0, 0)$ for $\ell = 4$, $(0, 0, 1, 0)$ for $\ell = 3$, $(0, 1, 0, 0)$ for $\ell = 2$

Number of steps:

- DP2 executes $S(n) = \dfrac{7n^2 + 13n + 6}{2} = \Theta(n^2)$ steps in the worst case

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with the RCP and its DP algorithms
- One concerns the counting of the solutions, one the base case, one the second recurrence, and one the reconstruction of a solution
- Check them before running the DP algorithms by hand

Typical mistakes (1/2)

Observation 1: Counting vectors instead of cutting patterns

The 2^{n-1} cutting patterns of a rod of length n are more than the solution vectors when $n \geq 3$, because different patterns can give the same vector.

Observation 2: Forgetting the base case

$v(0) = 0$ is used when the first piece has the length of the whole rod: $v(\ell)$ contains the term $p_\ell + v(0)$.

Typical mistakes (2/2)

Observation 3: Using row $j-1$ in the second recurrence

The term $p_j + v(j, \ell - j)$ stays in row j: otherwise a piece length could be cut only once.

Observation 4: Stopping the reconstruction early

The rule $x_{s[\ell]} \leftarrow x_{s[\ell]} + 1$, $\ell \leftarrow \ell - s[\ell]$ is repeated until $\ell = 0$; the last piece completes the rod.

Summary

Solutions and base case:

- different cutting patterns can give the same vector: count them separately
- $v(0) = 0$ is needed for the term $p_\ell + v(0)$ of the whole rod

Second recurrence and reconstruction:

- the term $p_j + v(j, \ell - j)$ stays in row j: a piece length can be cut many times
- the reconstruction goes on until the residual length is 0

In summary

Introduction: In summary

- We collect the key ideas of the chapter on the DP algorithms for the RCP
- The dashboard gathers the result on the reference instance of the chapter

Summary (1/2)

- The RCP asks how to cut a rod of length n into pieces, with each length available any number of times, to maximize the total profit.
- The recurrence $v(\ell) = \max_{j \leq \ell}\{p_j + v(\ell - j)\}$ has the optimal substructure and overlapping subproblems, and $v(0) = 0$.
- The vector v is filled bottom-up in $\Theta(n^2)$ steps, which is polynomial because the input has n profits.
- The vector s of the first pieces gives an optimal solution for every rod length at no extra cost.

Summary (2/2)

Dashboard of the RCP example (the reference instance, rod of length 4):

Method	Lower bound	Upper bound	Origin
DP algorithm	10	10	optimal value, two pieces of length 2