DP for the subset sum problem

Outline

A DP algorithm
    Number of steps

Typical mistakes

In summary

Learning objectives

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

- write the recurrence $r(j, k) \in \{true, false\}$ of the reachable sums, with its base cases;
- run the DP algorithm by hand and read the answer of the SSP instance in $r(n, c)$;
- fill the matrix s and reconstruct a subset of total weight c when the answer is yes;
- relate $r(j, k)$ to the DP algorithm of the KP with $p_j = w_j$, and count the $\Theta(nc)$ steps.

Prerequisites

- From this module, the chapter «DP for the 0/1 knapsack problem»: the recurrence $v(j, k)$ of the 0/1 knapsack problem (KP) over items and capacities, the matrix s of the decisions and the reconstruction of a solution.
- From the module *Combinatorial Optimization Problems*, the chapter «Knapsack problems»: the subset sum problem and its solution as a 0/1 knapsack problem with $p_j = w_j$.
- From the module *Algorithms*: pseudocode conventions and the size of the input.

The problem of this chapter

The problem of this chapter is the
subset sum problem (SSP):

its definition and its assumptions are on its page.

- We write a DP algorithm that gives the answer of an SSP instance, yes or no
- When the answer is yes, the same algorithm also gives a subset of total weight c

A DP algorithm

Introduction: A DP algorithm

- In this section we write a DP algorithm for the SSP, in two steps: first the answer of the instance, then a subset of total weight c
- The algorithm follows the DP algorithm of the KP, but each cell of its table stores only one bit, *true* or *false*
- We run it by hand on an instance with four items, and we count its steps

Example: SSP instance

We consider a target value $c = 10$ and a set $\mathscr{I} = \{1, 2, 3, 4\}$ of 4 items ($n = 4$), with the following weights:

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

We use this SSP instance as the example of the whole chapter.

Subsets and answers

- A subset $S \subseteq \mathscr{I}$ is described by the binary vector $(x_1, x_2, \ldots, x_n)$, where $x_j = 1$ if item j belongs to S and $x_j = 0$ otherwise.

- The answer to the SSP instance is **yes** if some vector satisfies

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

and **no** otherwise.

The two steps of the DP algorithm

- Step 1: computation of the answer
  - a recursive formula tells, for each subinstance, whether its target value can be reached
- Step 2: computation of a subset
  - when the answer is yes, a matrix of decisions gives a subset of total weight c

Step 1: the subinstances

For any $j \in \{0, 1, \ldots, n\}$ and $k \in \{0, 1, \ldots, c\}$, let

$$r(j, k) \in \{true, false\}$$

be the answer of the SSP subinstance given by:

1. a target value k,
2. a set $\{1, 2, \ldots, j\}$ of available items,
3. for each item $h \in \{1, 2, \ldots, j\}$, a weight w_h.

Step 1: the recursive formula

The recursive formula to compute $r(j,k)$ reads as follows:

$$r(j,k) = \begin{cases} \mathit{true} & \text{if } k = 0, \\ \mathit{false} & \text{if } j = 0 \text{ and } k \geq 1, \\ r(j-1,k) & \text{if } j,k \geq 1 \text{ and } w_j > k, \\ r(j-1,k) \vee r(j-1,k-w_j) & \text{if } j,k \geq 1 \text{ and } w_j \leq k. \end{cases}$$

Reading the recursive formula

- The symbol $\vee$ is the logical **or**: $r(j, k)$ is *true* if at least one of the two values is *true*.
- The base cases are:
  - $r(j, 0) =$ *true*, since the empty subset has total weight 0;
  - $r(0, k) =$ *false* for $k \geq 1$, since no item is available.

Optimal substructure

- A subset of $\{1, 2, \ldots, j\}$ with total weight k either:
  - does not contain item j, and then it is a subset of $\{1, 2, \ldots, j-1\}$ with total weight k, or
  - contains item j, and then the other items form a subset of $\{1, 2, \ldots, j-1\}$ with total weight $k - w_j$.

- Hence, the recursive formula generates at most 2 **independent subproblems**, and a subset for (j, k) is built from a subset of one of them by adding item j or not.

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

Overlapping subproblems and the bottom-up approach

- Different decisions on the items can lead to the same pair (j, k), and therefore the same value $r(j, k)$ may be needed several times during the recursive evaluation of the formula.

$$\rightsquigarrow \textbf{Overlapping Subproblems verified!}$$

- The value $r(j, k)$ depends only on values $r(j-1, k')$ with $k' \leq k$.
- Therefore, the **bottom-up approach** computes the rows $j = 0, 1, \ldots, n$ in this order, each one for increasing k, and stores every value in a **lookup table**.

The SSP and the DP algorithm of the KP

- The SSP can also be solved by the DP algorithm of the KP with $p_j = w_j$ for every item j.
- Its value $v(j, k)$ is the largest total weight, not larger than k, of a subset of $\{1, 2, \ldots, j\}$, and

$$r(j, k) = \mathit{true} \iff v(j, k) = k.$$

- The recurrence for $r(j, k)$ stores only one bit per cell, and its logical operations replace the sums and the comparisons of the KP recurrence.

Step 2: computation of a subset

- When the answer is yes, a subset of total weight c is reconstructed from the matrix s:
  - $s[j][k] = 1$ if item j is used to reach the value *true* of $r[j][k]$,
  - $s[j][k] = 0$ otherwise.
- The DP algorithm prefers not to use item j: $s[j][k] = 1$ only if $r[j-1][k]$ is *false* and $r[j-1][k-w_j]$ is *true*.

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**,
which stores the matrix s together with the matrix r.

Algorithm: SSP_DP (1/2)

Input : An SSP instance given by n, the array $w[1..n]$, and the target value c

Output: The answer $r[n][c]$ of the SSP instance and the matrix $s[0..n][0..c]$

```
1 function SSP_DP(n, w, c)
2 |   r := new array[0..n][0..c]; s := new array[0..n][0..c]
3 |   r[0][0] ← true; s[0][0] ← 0
4 |   for k := 1 to c do
5 |   |   r[0][k] ← false; s[0][k] ← 0
6 |   end
```

Algorithm: SSP_DP (2/2)

function *SSP_DP*(n, w, c) (continued)

```
 7 |  for j := 1 to n do
 8 |  |  for k := 0 to c do
 9 |  |  |  r[j][k] ← r[j − 1][k]; s[j][k] ← 0
10 |  |  |  if r[j][k] = false and w[j] ≤ k and
   |  |  |    r[j − 1][k − w[j]] = true then
11 |  |  |  |  r[j][k] ← true; s[j][k] ← 1
12 |  |  |  end
13 |  |  end
14 |  end
15 |  return r[n][c], s
16 end
```

How SSP_DP works

- The matrix $r[0..n][0..c]$ is the **lookup table** of the answers, and the matrix $s[0..n][0..c]$ stores the decisions.
- Lines 3–5 fill the row $j = 0$: only the target value 0 is reachable without items.
- For each pair (j, k), the DP algorithm first copies the answer without item j, and then uses item j only if this answer is *false* and item j makes the value k reachable.
- Finally, the answer of the SSP instance is returned as $r[n][c]$, together with the matrix s.

Example: execution of the DP algorithm (1/9)

- We consider the SSP instance of the example above, with target value $c = 10$ and weights $w = (3, 5, 4, 7)$.
- For each step of the DP algorithm with $j \geq 1$, the next tables show:
  - the values of j and k,
  - the cells $r[j][k]$ and $s[j][k]$,
  - the operation performed;

  T stands for *true* and F for *false*.
- The row $j = 0$ is $r[0][0] = \mathrm{T}$ and $r[0][k] = \mathrm{F}$ for $k \geq 1$, with $s[0][k] = 0$.

Example: execution of the DP algorithm (2/9)

j	k	$r[j][k]$	$s[j][k]$	Action
1	0	T	0	$w_1 = 3 > 0$: $r[0][0] = \mathrm{T}$
1	1	F	0	$w_1 = 3 > 1$: $r[0][1] = \mathrm{F}$
1	2	F	0	$w_1 = 3 > 2$: $r[0][2] = \mathrm{F}$
1	3	T	1	$r[0][3] \vee r[0][0] = \mathrm{F} \vee \mathrm{T} = \mathrm{T}$
1	4	F	0	$r[0][4] \vee r[0][1] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
1	5	F	0	$r[0][5] \vee r[0][2] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
1	6	F	0	$r[0][6] \vee r[0][3] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
1	7	F	0	$r[0][7] \vee r[0][4] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
1	8	F	0	$r[0][8] \vee r[0][5] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
1	9	F	0	$r[0][9] \vee r[0][6] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
1	10	F	0	$r[0][10] \vee r[0][7] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$

Example: execution of the DP algorithm (3/9)

j	k	$r[j][k]$	$s[j][k]$	Action
2	0	T	0	$w_2 = 5 > 0$: $r[1][0] = \mathrm{T}$
2	1	F	0	$w_2 = 5 > 1$: $r[1][1] = \mathrm{F}$
2	2	F	0	$w_2 = 5 > 2$: $r[1][2] = \mathrm{F}$
2	3	T	0	$w_2 = 5 > 3$: $r[1][3] = \mathrm{T}$
2	4	F	0	$w_2 = 5 > 4$: $r[1][4] = \mathrm{F}$
2	5	T	1	$r[1][5] \vee r[1][0] = \mathrm{F} \vee \mathrm{T} = \mathrm{T}$
2	6	F	0	$r[1][6] \vee r[1][1] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
2	7	F	0	$r[1][7] \vee r[1][2] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
2	8	T	1	$r[1][8] \vee r[1][3] = \mathrm{F} \vee \mathrm{T} = \mathrm{T}$
2	9	F	0	$r[1][9] \vee r[1][4] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
2	10	F	0	$r[1][10] \vee r[1][5] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$

Example: execution of the DP algorithm (4/9)

j	k	$r[j][k]$	$s[j][k]$	Action
3	0	T	0	$w_3 = 4 > 0$: $r[2][0] = \mathrm{T}$
3	1	F	0	$w_3 = 4 > 1$: $r[2][1] = \mathrm{F}$
3	2	F	0	$w_3 = 4 > 2$: $r[2][2] = \mathrm{F}$
3	3	T	0	$w_3 = 4 > 3$: $r[2][3] = \mathrm{T}$
3	4	T	1	$r[2][4] \vee r[2][0] = \mathrm{F} \vee \mathrm{T} = \mathrm{T}$
3	5	T	0	$r[2][5] \vee r[2][1] = \mathrm{T} \vee \mathrm{F} = \mathrm{T}$
3	6	F	0	$r[2][6] \vee r[2][2] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$
3	7	T	1	$r[2][7] \vee r[2][3] = \mathrm{F} \vee \mathrm{T} = \mathrm{T}$
3	8	T	0	$r[2][8] \vee r[2][4] = \mathrm{T} \vee \mathrm{F} = \mathrm{T}$
3	9	T	1	$r[2][9] \vee r[2][5] = \mathrm{F} \vee \mathrm{T} = \mathrm{T}$
3	10	F	0	$r[2][10] \vee r[2][6] = \mathrm{F} \vee \mathrm{F} = \mathrm{F}$

Example: execution of the DP algorithm (5/9)

j	k	$r[j][k]$	$s[j][k]$	Action
4	0	T	0	$w_4 = 7 > 0$: $r[3][0] = \mathrm{T}$
4	1	F	0	$w_4 = 7 > 1$: $r[3][1] = \mathrm{F}$
4	2	F	0	$w_4 = 7 > 2$: $r[3][2] = \mathrm{F}$
4	3	T	0	$w_4 = 7 > 3$: $r[3][3] = \mathrm{T}$
4	4	T	0	$w_4 = 7 > 4$: $r[3][4] = \mathrm{T}$
4	5	T	0	$w_4 = 7 > 5$: $r[3][5] = \mathrm{T}$
4	6	F	0	$w_4 = 7 > 6$: $r[3][6] = \mathrm{F}$
4	7	T	0	$r[3][7] \vee r[3][0] = \mathrm{T} \vee \mathrm{T} = \mathrm{T}$
4	8	T	0	$r[3][8] \vee r[3][1] = \mathrm{T} \vee \mathrm{F} = \mathrm{T}$
4	9	T	0	$r[3][9] \vee r[3][2] = \mathrm{T} \vee \mathrm{F} = \mathrm{T}$
4	10	T	1	$r[3][10] \vee r[3][3] = \mathrm{F} \vee \mathrm{T} = \mathrm{T}$

Example: execution of the DP algorithm (6/9)

This is the matrix r after the execution of the DP algorithm, columns $k = 0, 1, \ldots, 5$:

$r[j][k]$	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$j=0$	T	F	F	F	F	F
$j=1$	T	F	F	T	F	F
$j=2$	T	F	F	T	F	T
$j=3$	T	F	F	T	T	T
$j=4$	T	F	F	T	T	T

Example: execution of the DP algorithm (7/9)

This is the matrix r after the execution of the DP algorithm, columns $k = 6, 7, \ldots, 10$:

$r[j][k]$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
$j = 0$	F	F	F	F	F
$j = 1$	F	F	F	F	F
$j = 2$	F	F	T	F	F
$j = 3$	F	T	T	T	F
$j = 4$	F	T	T	T	T

Example: execution of the DP algorithm (8/9)

This is the matrix s after the execution of the DP algorithm, columns $k = 0, 1, \ldots, 5$:

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

Example: execution of the DP algorithm (9/9)

This is the matrix s after the execution of the DP algorithm, columns $k = 6, 7, \ldots, 10$:

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

- Since $r[4][10] = \mathrm{T}$, the answer to the SSP instance is **yes**.
- For example, $s[4][10] = 1$ since $r[3][10] = \mathrm{F}$ and $r[3][3] = \mathrm{T}$: item 4 is needed to reach the target value 10.

Reconstruction of a subset

- Once the matrix s has been computed and $r[n][c]$ is *true*, a subset of total weight c is reconstructed:
  - scanning the items from $j = n$ down to $j = 1$,
  - starting from $k = c$ and $x_j = 0$ for every item j:

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

- The reconstruction ends with $k = 0$, since every step keeps $r[j-1][k]$ equal to *true*.

Algorithm: SSP_Reconstruct

Input : The number n of items, the target value c, the array $w[1..n]$, and the matrix $s[0..n][0..c]$, with $r[n][c] = true$

Output: A subset of total weight c, as a binary vector

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

Example: computing a subset

After the execution of the DP algorithm on the instance of the example above, we start from $j = 4$ and $k = 10$:

Iteration	j	k	$s[j][k]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	Action
–	–	–	–	0	0	0	0	$k = 10$
1	4	10	1	0	0	0	1	$x[4] \leftarrow 1,\ k \leftarrow 10 - 7 = 3$
2	3	3	0	0	0	0	1	$x[3] \leftarrow 0,\ k = 3$
3	2	3	0	0	0	0	1	$x[2] \leftarrow 0,\ k = 3$
4	1	3	1	1	0	0	1	$x[1] \leftarrow 1,\ k \leftarrow 3 - 3 = 0$

The answer to the SSP instance of the example is **yes**: the vector $(1, 0, 0, 1)$ selects the subset $\{1, 4\}$, whose total weight is $3 + 7 = c$.

How many steps does the DP algorithm execute?

We count how many times each line
of the pseudo-code of SSP_DP is executed.

Number of steps (1/2)

Let $S(n, c)$ denote the number of steps executed by `SSP_DP` in the worst case. We count how many times each line of the pseudo-code is executed, assuming that the condition at line 10 is always true:

- Lines 2 and 3 are executed 1 time each.
- Line 4 is executed c times, and line 5 c times.
- Line 7 is executed n times.
- Lines 8, 9, 10 and 11 are executed $n(c + 1)$ times each.
- Line 15 is executed 1 time.

Number of steps (2/2)

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

$$S(n,c) = 1 + 1 + 2c + n + 4n(c+1) + 1 = 4n(c+1) + n + 2c + 3.$$

Hence,

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

since

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

$$= 4 \in \mathbb{R}_{>0}.$$

The number of steps of the DP algorithm

The DP algorithm for the SSP executes

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

steps in the worst case,
the same number as the DP algorithm of the KP.

- The reconstruction executes $\Theta(n)$ steps, one for each item.
- The number of steps contains the target value c, a number of the input, and not only the size of the input: it is **pseudo-polynomial**.

Summary (1/2)

Step 1: the answer:

- $r(j,k) = r(j-1,k) \vee r(j-1,k-w_j)$ if $w_j \leq k$, and $r(j,k) = r(j-1,k)$ if $w_j > k$
- base cases $r(j,0) = \mathit{true}$ and $r(0,k) = \mathit{false}$ for $k \geq 1$; the answer is $r(n,c)$
- optimal substructure and overlapping subproblems: bottom-up, row by row, in a lookup table

Relation with the KP:

- with $p_j = w_j$, $r(j,k) = \mathit{true}$ if and only if $v(j,k) = k$
- one bit per cell, logical operations instead of sums and comparisons

Summary (2/2)

Step 2: a subset:

- $s[j][k] = 1$ only if $r[j-1][k]$ is *false* and $r[j-1][k-w_j]$ is *true*
- the subset is read backwards from (n, c): $x_j \leftarrow s[j][k]$, $k \leftarrow k - s[j][k]\, w_j$
- on the example: the subset $\{1, 4\}$, with total weight $3 + 7 = 10$

Number of steps:

- $S(n, c) = \Theta(nc)$ for SSP_DP, $\Theta(n)$ for the reconstruction
- the number of steps depends on the value c: it is pseudo-polynomial

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with the DP algorithm for the SSP
- Two of them concern the base cases, two the answer and the reconstruction
- Check them before running the DP algorithm by hand

Typical mistakes (1/2)

Observation 1: Setting $r(j, 0) = \textit{false}$

The target value 0 *is always reachable with the empty subset; with* $r(j, 0) = \textit{false}$ *no value would ever become true.*

Observation 2: Setting $r(0, k) = \textit{true}$ for $k \geq 1$

Without items, only the target value 0 *is reachable.*

Typical mistakes (2/2)

Observation 3: Reading the answer in the wrong cell

The answer is $r(n, c)$; a true value in another cell of the last row says that a different target value is reachable.

Observation 4: Reconstructing when the answer is no

If $r(n, c)$ is false, no subset has total weight c, and the matrix s gives nothing to reconstruct.

Summary

Base cases:

- $r(j, 0) = \mathit{true}$: the empty subset has total weight 0
- $r(0, k) = \mathit{false}$ for $k \geq 1$: without items only 0 is reachable

Answer and reconstruction:

- the answer is in the cell $r(n, c)$, not in another cell of the last row
- reconstruct a subset only when $r(n, c)$ is *true*

In summary

Introduction: In summary

- We collect the key ideas of the chapter on the DP algorithm for the SSP
- The dashboard gathers the result on the example of the chapter

Summary (1/2)

- The recurrence $r(j,k) = r(j-1,k) \vee r(j-1,k-w_j)$ tells whether the target value k is reachable with the items $1, 2, \ldots, j$.
- The base cases are $r(j,0) = \mathit{true}$ and $r(0,k) = \mathit{false}$ for $k \geq 1$; the answer is $r(n,c)$.
- The matrix s stores whether item j is used, and a subset of total weight c is read backwards from (n,c).
- The DP algorithm executes $\Theta(nc)$ steps, as the DP algorithm of the KP with $p_j = w_j$.

Summary (2/2)

Dashboard of the SSP example ($c = 10$, four items):

Method	Answer	Origin
DP algorithm	yes	$r(4, 10) = \mathit{true}$, subset $\{1, 4\}$