ALGORITHMS AND DATA STRUCTURES · LECTURE 18

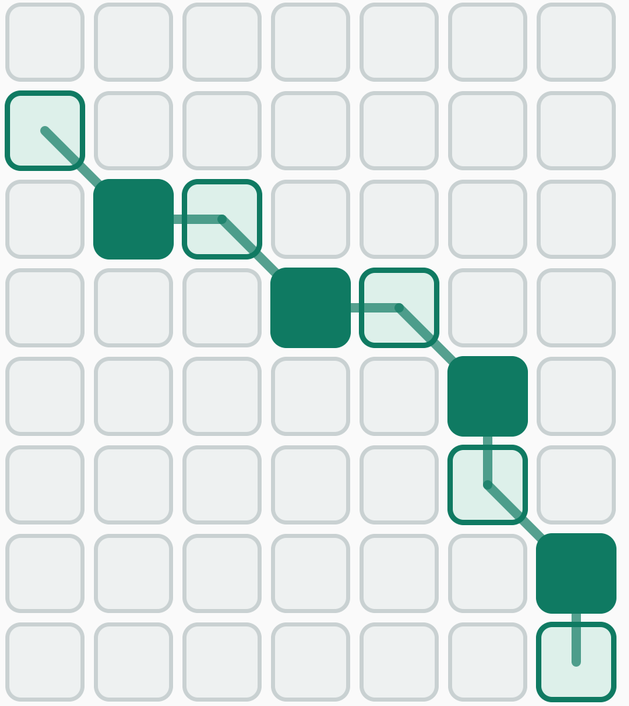
Longest common subsequence and subset sum

Dynamic programming

Heng Guo

School of Informatics, University of Edinburgh

Thursday 19 November 2026 · 16:10–17:00

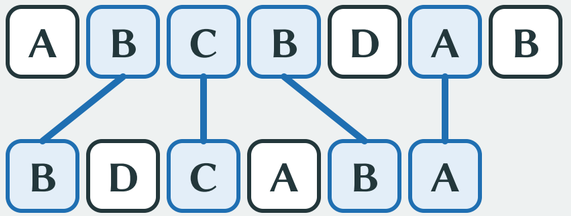


The recipe from last time

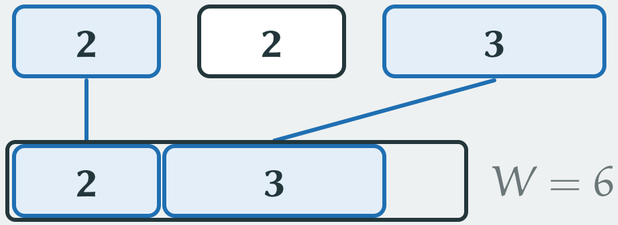
1. **Subproblems**
   smaller versions of the problem
2. **Optimal substructure**
   an optimal solution contains
   optimal solutions of subproblems
3. **Recurrence**
   the value from smaller subproblems; base cases
4. **Table**
   store the answers; fill in dependency order (or memoise)
5. **Traceback**
   keep the choices, read the solution off the table

Lecture 17: chain matrix multiplication, $O(n^3)$.

Strings: longest common subsequence



Numbers: subset sum



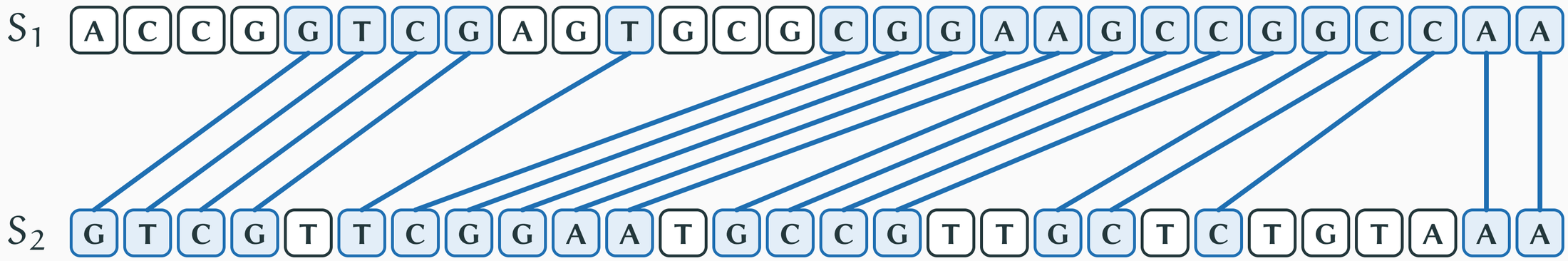
Longest common subsequence

How similar are two strings?

Comparing two strands

Two DNA strands S_1, S_2: strings over the letters **A C G T**.

? *How similar are they?*



A **common subsequence**: letters that occur in both, in the same order,
not necessarily next to each other.

Here: **GTCGTCGGAAGCCGGCCAA**, 19 letters.

How do we find a longest one?

Common subsequences

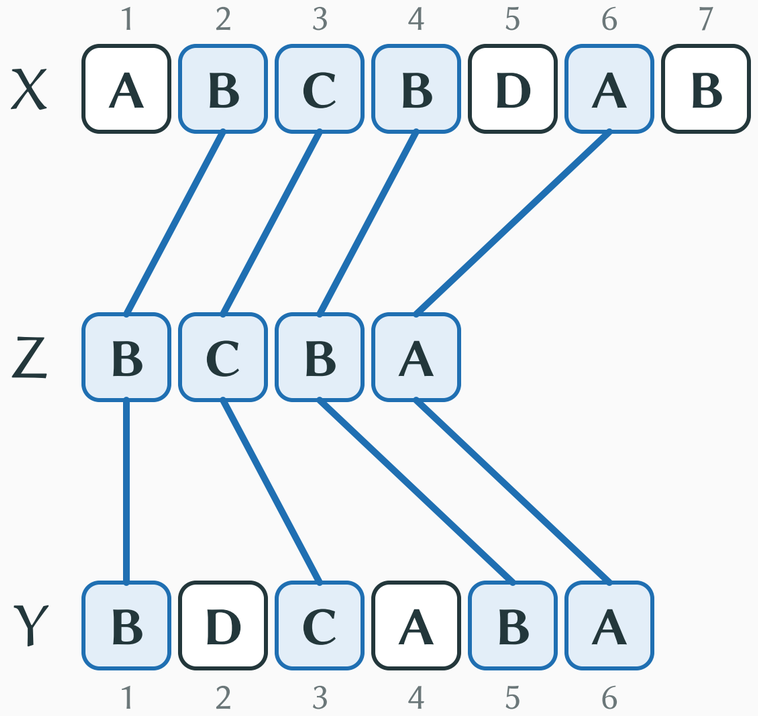
Subsequence.
Keep some letters of $X = \langle x_1, \ldots, x_n \rangle$, in order:
$Z = \langle x_{i_1}, \ldots, x_{i_k} \rangle$ with $i_1 < i_2 < \cdots < i_k$.

BCDB is a subsequence of X:
positions $2 < 3 < 5 < 7$.

A **common subsequence** of X and Y is
a subsequence of both, for example **BCA**.

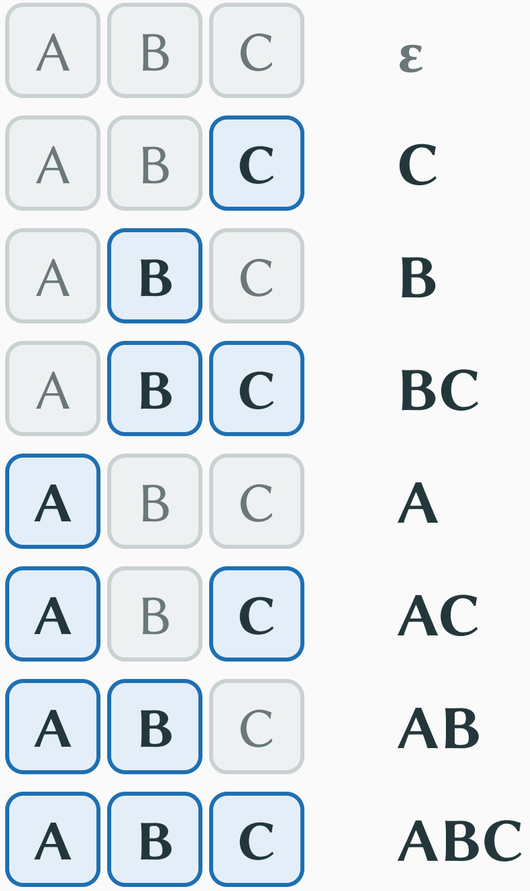
Longest common subsequence (LCS).
Given $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$,
find a common subsequence of maximum length.

Here **BCBA**, of length 4.



Brute force

$X = ABC$: kept blue, dropped grey



? *How many subsequences does X have?*

Keep or drop each of the n letters:
2^n subsets of the positions.

Test each against Y by one scan, $O(m+n)$:
time $O\big((m+n)\,2^n\big)$.

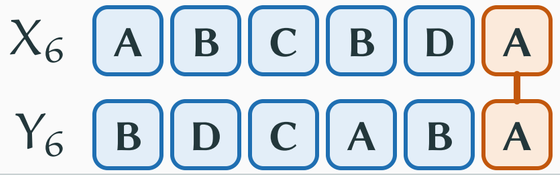
> $n = 50$: $2^{50} > 10^{15}$ candidates. **Exponential.**

Optimal substructure

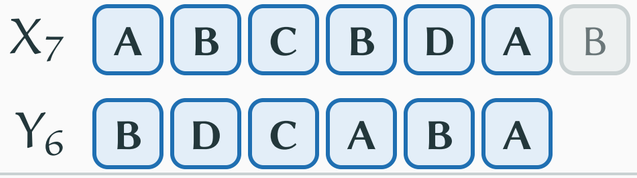
Subproblems: the prefixes $X_i = \langle x_1, \ldots, x_i \rangle$ and $Y_j = \langle y_1, \ldots, y_j \rangle$,
for $0 \leqslant i \leqslant n$, $0 \leqslant j \leqslant m$ (X_0 is empty, $X_n = X$, $Y_m = Y$).

Optimal substructure. Let $Z = \langle z_1, \ldots, z_k \rangle$ be **any** LCS of X_n and Y_m. Then:

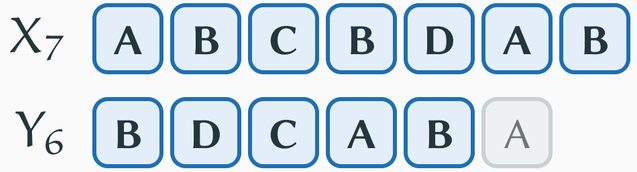
(1) $x_n = y_m$: then $z_k = x_n = y_m$,
and Z_{k-1} is an LCS of X_{n-1} and Y_{m-1}.



(2) $x_n \neq y_m$ and $z_k \neq x_n$: then Z is an LCS of X_{n-1} and Y_m.



(3) $x_n \neq y_m$ and $z_k \neq y_m$: then Z is an LCS of X_n and Y_{m-1}.



If $x_n \neq y_m$, then z_k differs from x_n or from y_m: **(2) or (3) applies**, and we do not know which.

Proof: cut and paste

Each claim by contradiction:
Z is **longest**, so no longer common subsequence exists.

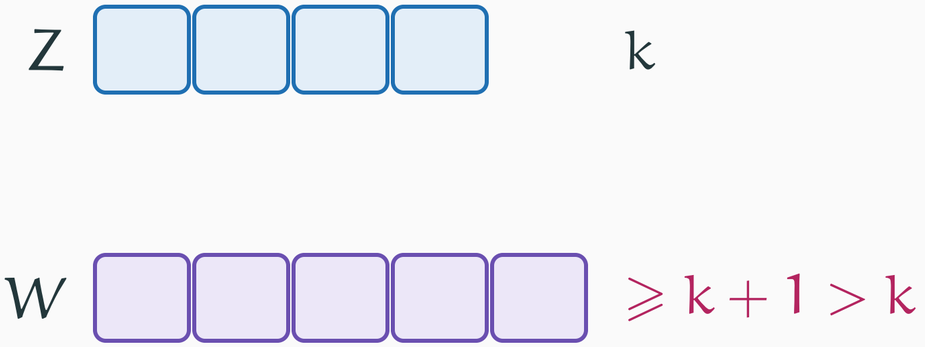
(1) $x_n = y_m$. If $z_k \neq x_n$, appending x_n to Z gives a common subsequence of length $k+1$.
Hence $z_k = x_n = y_m$.

Z_{k-1} is common to X_{n-1}, Y_{m-1}.
If W were a longer one, $|W| \geqslant k$, then
W followed by x_n is common to X_n, Y_m,
with length $\geqslant k+1$.

(2) $x_n \neq y_m$, $z_k \neq x_n$. Then Z is common to X_{n-1}, Y_m.
A longer one would be common to X_n, Y_m.

(3) follows by exchanging X and Y. □

Claim (3). If $x_n \neq y_m$ and $z_k \neq y_m$, then Z is an LCS of X_n and Y_{m-1}.



The recurrence

$c[i, j]$ is the length of an LCS of X_i and Y_j.
We want $c[n, m]$.

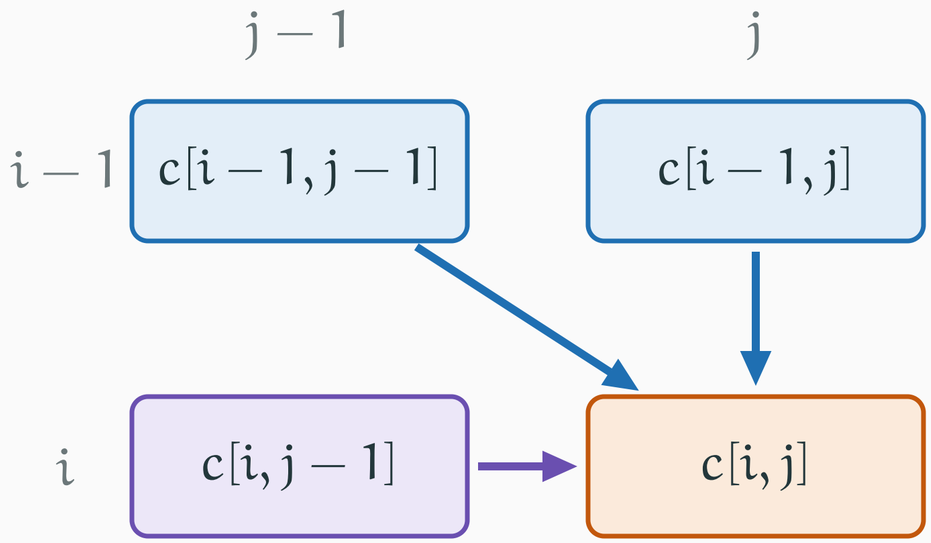
$i = 0$ or $j = 0$: $\quad c[i, j] = 0$

$x_i = y_j$: $\quad c[i, j] = c[i-1, j-1] + 1$

Claim (1)

$x_i \neq y_j$: $\quad c[i, j] = \max\big(c[i-1, j],\ c[i, j-1]\big)$

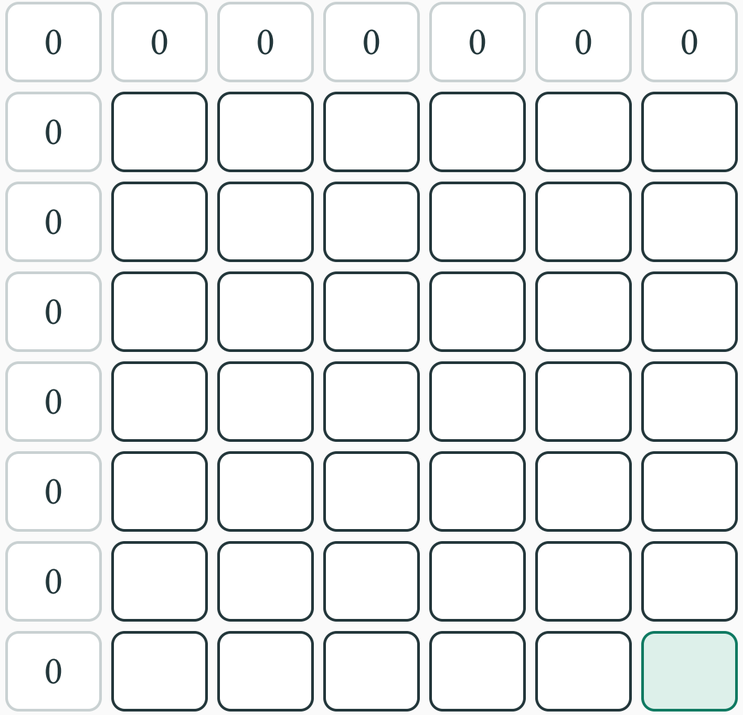
Claims (2), (3)



$(n+1)(m+1)$ subproblems, each from three neighbours in $O(1)$ time: **fill the table row by row.**

The algorithm

```
LCS-Length(X, Y, n, m)
1  let c[0..n, 0..m] be a new table
2  for i = 0 to n do c[i, 0] ← 0
3  for j = 0 to m do c[0, j] ← 0
4  for i = 1 to n do
5      for j = 1 to m do
6          if x_i = y_j then
7              c[i, j] ← c[i − 1, j − 1] + 1
8          else c[i, j] ← max(c[i − 1, j], c[i, j − 1])
9  return c[n, m]
```



Time: nm entries, $O(1)$ each: $\Theta(nm)$.

Space: $\Theta(nm)$.

Two rows suffice for the length alone.

Worked example

i\j	0	1 B	2 D	3 C	4 A	5 B	6 A
0	0	0	0	0	0	0	0
1 A	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖
2 B	0	1 ↖	1 ←	1 ←	1 ↑	2 ↖	2 ←
3 C	0	1 ↑	1 ↑	2 ↖	2 ←	2 ↑	2 ↑
4 B	0	1 ↖	1 ↑	2 ↑	2 ↑	3 ↖	3 ←
5 D	0	1 ↑	2 ↖	2 ↑	2 ↑	3 ↑	3 ↑
6 A	0	1 ↑	2 ↑	2 ↑	3 ↖	3 ↑	4 ↖
7 B	0	1 ↖	2 ↑	2 ↑	3 ↑	4 ↖	4 ↑

$X = \mathbf{ABCBDAB}, \quad Y = \mathbf{BDCABA}$

$c[7, 6] = 4$: the length of a longest common subsequence.

- entry being computed
- entry it uses: diagonal (match), or above
- entry it uses: left
- ↖ ↑ ← where the value came from

Recover an optimal subsequence

i\j	0	1 B	2 D	3 C	4 A	5 B	6 A
0	0	0	0	0	0	0	0
1 A	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖
2 B	0	1 ↖	1 ←	1 ←	1 ↑	2 ↖	2 ←
3 C	0	1 ↑	1 ↑	2 ↖	2 ←	2 ↑	2 ↑
4 B	0	1 ↖	1 ↑	2 ↑	2 ↑	3 ↖	3 ←
5 D	0	1 ↑	2 ↖	2 ↑	2 ↑	3 ↑	3 ↑
6 A	0	1 ↑	2 ↑	2 ↑	3 ↖	3 ↑	4 ↖
7 B	0	1 ↖	2 ↑	2 ↑	3 ↑	4 ↖	4 ↑

$c[7,6] = 4$: the traceback has collected an LCS of length 4.

the LCS, read backwards

B **C** **B** **A**

Tie at $c[7,6]$: going left (purple) gives another LCS, **BDAB**.

? *Can we use less space?*

Saving space: two rows

i\j	0	1 B	2 D	3 C	4 A	5 B	6 A
0							
1 A							
2 B							
3 C							
4 B							
5 D							
6 A	0	1	2	2	3	3	4
7 B	0	1	2	2	3	4	4

Each entry uses only the row above and its own row.

Keep only **two rows** in memory.

Row i: compute it from row $i-1$, then forget row $i-2$.

$c[7,6] = 4$ from $2(m+1)$ entries: space $O(m)$, time still $O(nm)$.

Recovering the subsequence itself in linear space needs more work (Hirschberg, 1975).

Subset sum

Which subset of the weights is largest without exceeding W?

The subset sum problem

Items $1, \ldots, n$ with weights $w_i \geqslant 0$,
and a bound W.

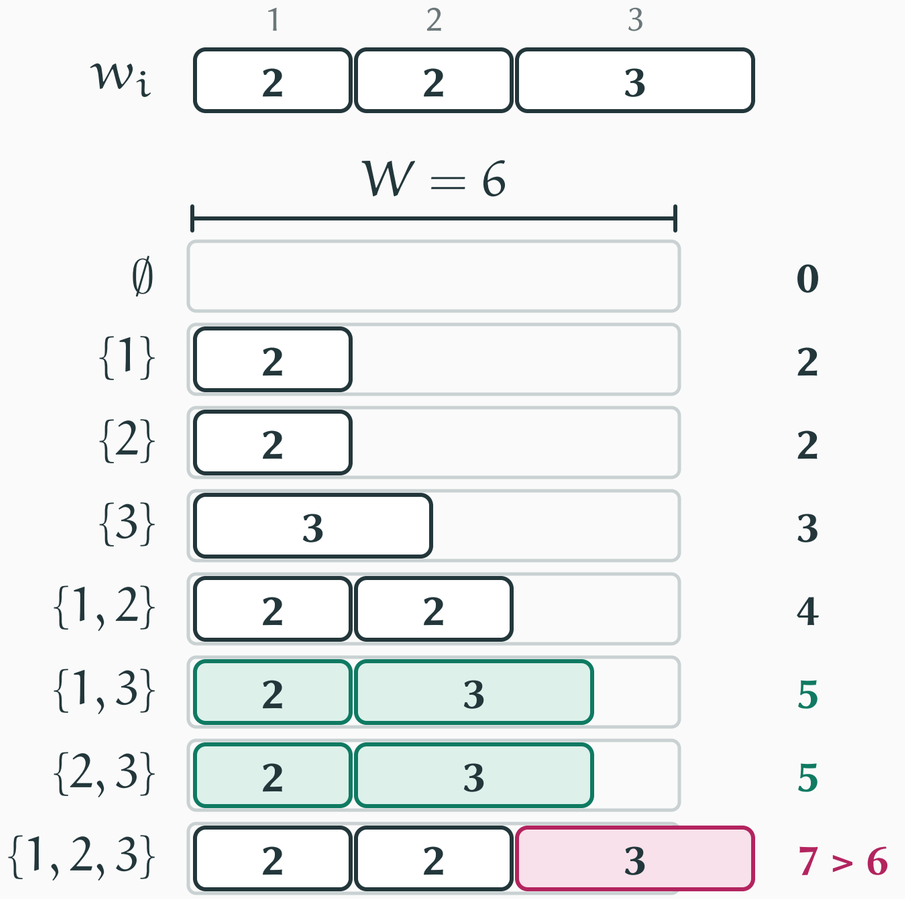
> **Subset sum.** Choose $S \subseteq \{1, \ldots, n\}$ with
> $\sum_{i \in S} w_i \leqslant W$ and
> $\sum_{i \in S} w_i$ **as large as possible**.

Example: $n = 3$, $W = 6$, $w = (2, 2, 3)$.

Best total weight 5, by $\{1, 3\}$ or by $\{2, 3\}$.

? *How many subsets does brute force try?*

2^n. Exponential again.



A first attempt

Try the first i items:
$\mathrm{OPT}(i)$ is the best total weight
using items $1, \ldots, i$ (bound W).

? *Is item n in an optimal solution?*

No: then $\mathrm{OPT}(n) = \mathrm{OPT}(n-1)$.

Yes: then $\mathrm{OPT}(n) = w_n + \mathrm{OPT}(n-1)$?

> The other items may weigh at most $W - w_n$.
> Use a **smaller bound** instead of $\mathrm{OPT}(n-1)$.

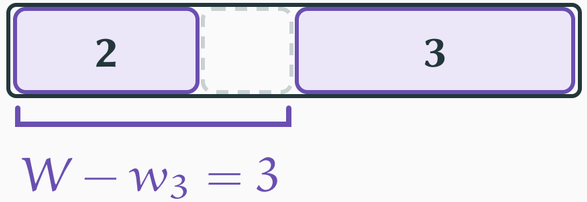
$\mathrm{OPT}(2) = 2 + 2 = 4$



$w_3 + \mathrm{OPT}(2) = 3 + 4 = 7 > 6$



$w_3 + 2 = 3 + 2 = 5$



The right subproblems

Subproblems.
$\mathrm{OPT}(i, w)$ is the largest total weight of a subset of $\{1, \ldots, i\}$ with total at most w, for $0 \leqslant i \leqslant n$ and $0 \leqslant w \leqslant W$.

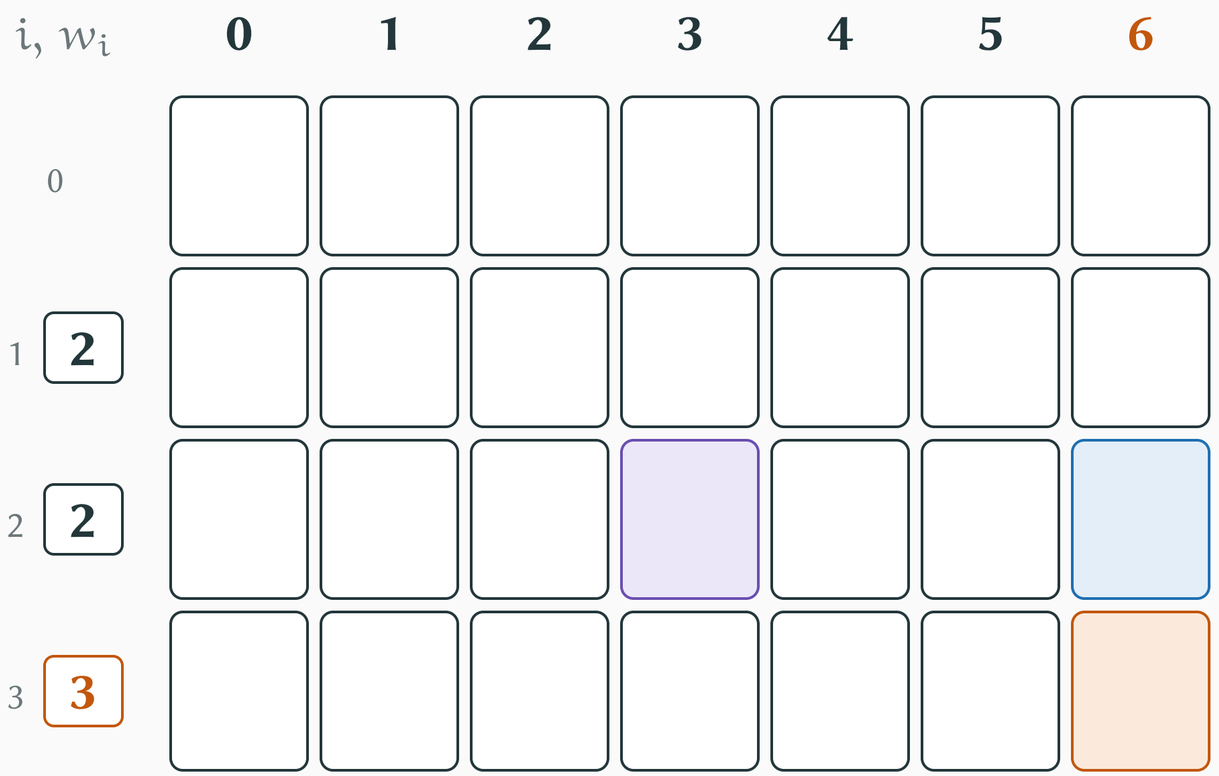
Assume that all w_i and W are integers.

We want $\mathrm{OPT}(n, W)$.

Item n out: $\mathrm{OPT}(n-1, W)$.

Item n in: $w_n + \mathrm{OPT}(n-1, W - w_n)$.

$(n+1)(W+1)$ subproblems, here $4 \cdot 7 = 28$.

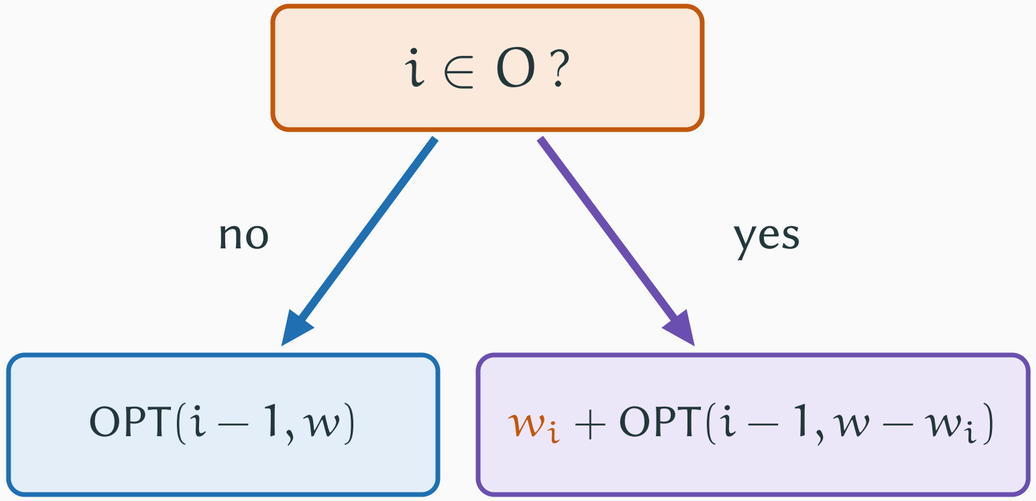


Optimal substructure and the recurrence

Let O be an optimal solution for (i, w).
Is item i in O?

No: O is a solution for $(i-1, w)$, and optimal there.

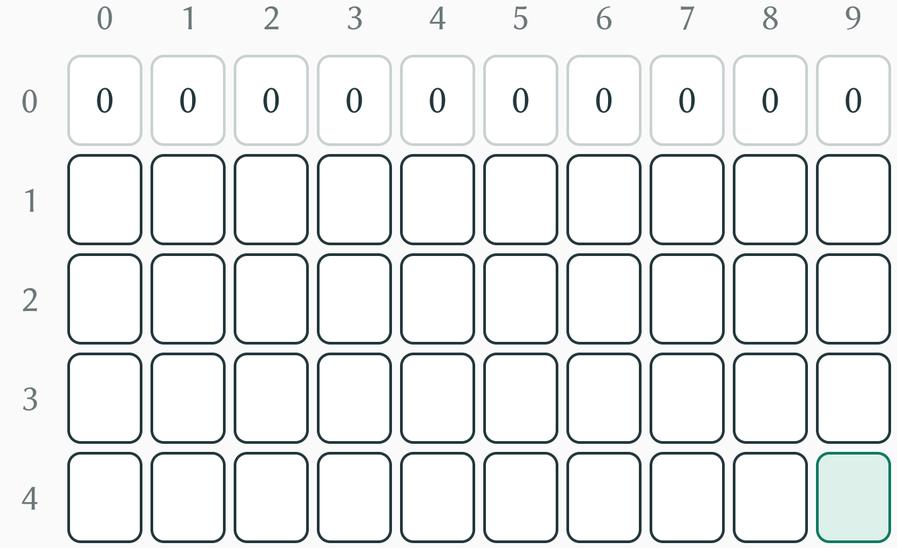
Yes: $O \setminus \{i\}$ is optimal for $(i-1, w-w_i)$.
If a better set S' existed,
$S' \cup \{i\}$ would beat O.



$$
\begin{aligned}
w_i > w\text{:} \quad & \mathrm{OPT}(i, w) = \mathrm{OPT}(i-1, w) \\
w_i \leqslant w\text{:} \quad & \mathrm{OPT}(i, w) = \max\big(\mathrm{OPT}(i-1, w),\ w_i + \mathrm{OPT}(i-1, w-w_i)\big) \\
i = 0\text{:} \quad & \mathrm{OPT}(0, w) = 0
\end{aligned}
$$

The algorithm

```
SUBSET-SUM(w_1, ..., w_n, W)
1  let M[0..n, 0..W] be a new table
2  for w = 0 to W do M[0, w] ← 0
3  for i = 1 to n do
4      for w = 0 to W do
5          if w_i > w then
6              M[i, w] ← M[i − 1, w]
7          else
8              M[i, w] ← max(M[i − 1, w], w_i + M[i − 1, w − w_i])
9  return M[n, W]
```



Entry (i, w) reads the row above: $(i-1, w)$ and $(i-1, w-w_i)$.

Worked example

i, w_i	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
1 **2**	0	0	2	2	2	2	2
2 **2**	0	0	2	2	4	4	4
3 **3**	0	0	2	3	4	5	5

$n = 3, W = 6, w = (2, 2, 3)$

$M[3, 6] = 5$: the best total weight for all three items and bound 6.

- entry being computed
- the entry above: item i left out
- w_i columns to the left, above: item i taken

From values to solutions

i, w_i	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
1 **2**	0	0	2	2	2	2	2
2 **2**	0	0	2	2	4	4	4
3 **3**	0	0	2	3	4	5	5

$M[3, 6] = 5$: the traceback has collected a solution of total weight 5.

$S = \{1, 3\}$

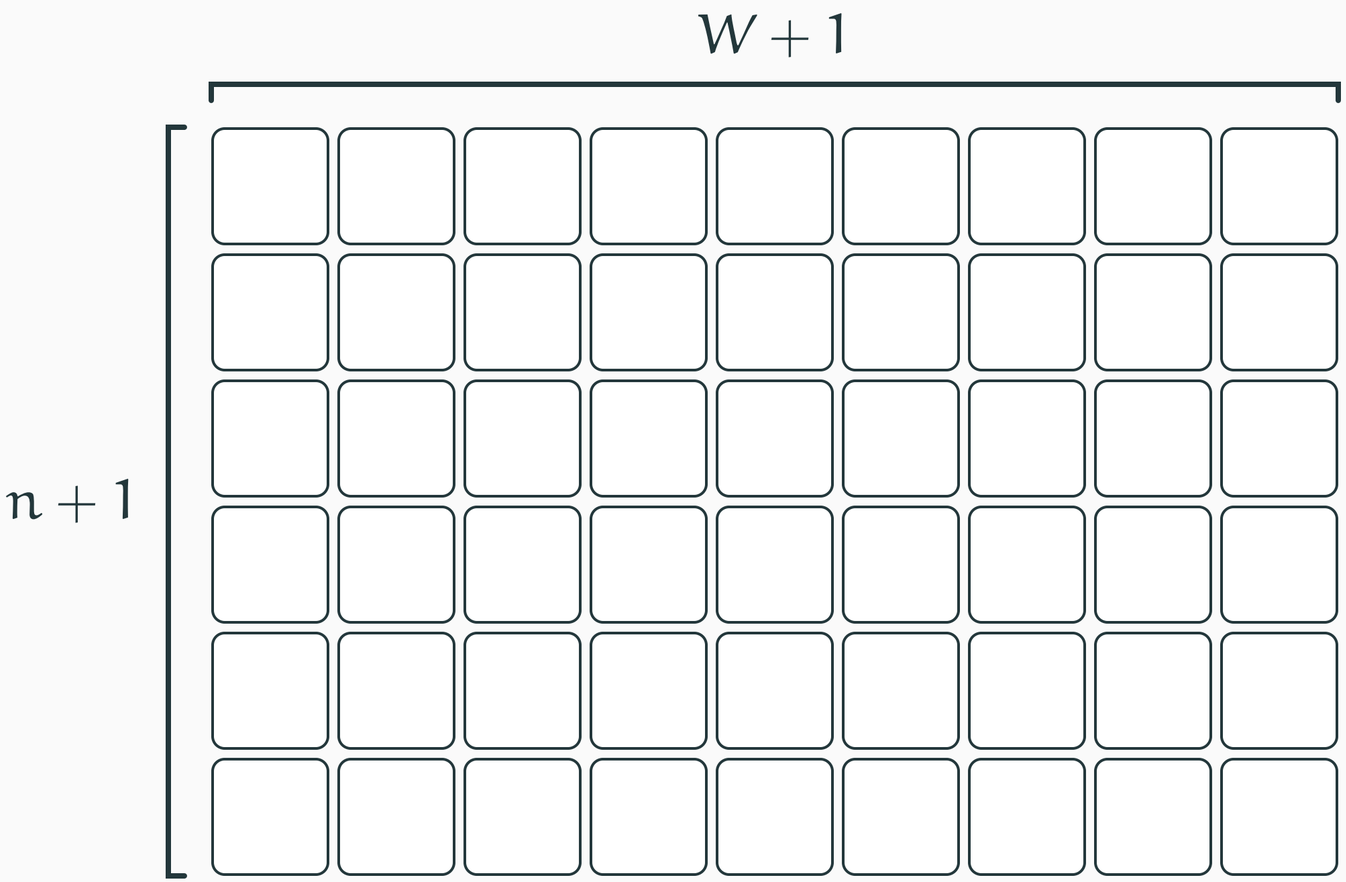


$S = \{2, 3\}$



Tie at $M[2, 3]$: taking item 2 instead gives $\{2, 3\}$, also optimal.

Running time



The table has $(n + 1)(W + 1)$ entries.

Each entry: two look-ups, $O(1)$ time.

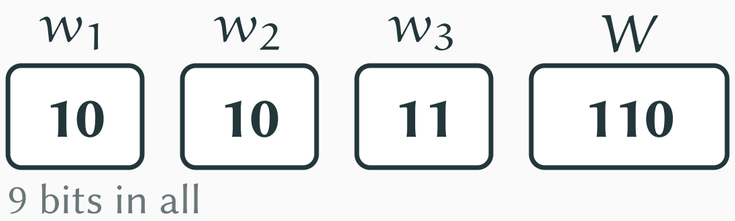
For $n = 100$, $W = 10^4$: 10^6 steps.

> Time $O(nW)$, space $\Theta(nW)$.
>
> The traceback takes $O(n)$ steps.

? *Is this a polynomial-time algorithm?*

The size of the input

The input: $w_1, \ldots, w_n$ and W, written in **binary**.



$n + 1$ numbers of $\log_2 W$ bits: size $N \approx n \log_2 W$.

$nW = n \cdot 2^{\log_2 W}$ is exponential in the number of bits of W.

bits of W	W	nW
10	10^3	10^5
20	10^6	10^8
30	10^9	10^{11}
40	10^{12}	10^{14}

$n = 100$ items.
Each extra bit of W doubles the table.

Pseudo-polynomial. Polynomial in n and in the **value** of W, not in its number of bits.

Should we be happy?

? *Is $O(nW)$ good news?*

Yes, when the numbers are small.

$n = 100$, $W = 10^4$ (14 bits):
10^6 entries, instant.

No, when they are large.

$n = 100$, $W = 10^{12}$ (40 bits):
10^{14} entries, from an input of about 4000 bits.

No polynomial-time algorithm is known. Subset sum is **NP-hard**:
a polynomial-time algorithm would settle P versus NP,
one of the Clay Millennium Problems. More in Lecture 19.

The recipe, twice

	longest common subsequence	subset sum
1 Subproblems	(X_i, Y_j): prefixes of both strings	(i, w): the first i items, bound w
2 Optimal substructure	the last letters are equal, or one is unused	item i out, or in: the bound drops by w_i
3 Recurrence	$c[i-1, j-1] + 1$ if the letters match, else $\max\big(c[i-1, j],\ c[i, j-1]\big)$	$\max\big(\mathrm{OPT}(i-1, w),\ w_i + \mathrm{OPT}(i-1, w-w_i)\big)$
4 Table	$(n+1)(m+1)$ entries: $O(nm)$	$(n+1)(W+1)$ entries: $O(nW)$
5 Traceback	follow the arrows, collect letters	compare with the entry above, collect items

Same recipe, different subproblems. Find the correct recurrence.

The best algorithms known

Longest common subsequence

two strings of length n

$O(n^2)$: this lecture.

$O(n^2 / \log n)$ by precomputing small blocks.

Masek and Paterson, 1980

No $O(n^{2-\varepsilon})$ algorithm unless SETH fails.

Abboud, Backurs and Vassilevska Williams, 2015

Bringmann and Künnemann, 2015

26 October 2026. OpenAI and Anthropic announced $2^{o(n)}$-time algorithms for 3-SAT (checked in Lean).
Hence **ETH is false**, and so is SETH.
No faster algorithm for LCS or subset sum yet.

Subset sum

n items, bound W

$O(nW)$: this lecture.

Bellman, 1957

$\tilde{O}(n + W)$, randomised.

Bringmann, 2017

No $O(W^{1-\varepsilon}\, 2^{o(n)})$ algorithm unless SETH fails.

Abboud, Bringmann, Hermelin and Shabtay, 2019

In terms of n only: $O(2^{n/2}\, n)$.

Horowitz and Sahni, 1974

SETH (Strong Exponential Time Hypothesis): satisfiability of formulas with n variables needs time roughly $\Omega(2^n)$.

ETH (Exponential Time Hypothesis, implied by SETH): 3-SAT, with three literals per clause, needs time $2^{\Omega(n)}$.

Summary

1. **Longest common subsequence:** look at the last letters;
   the table $c[i, j]$ of prefix pairs;
   traceback.

2. **Subset sum:** the subproblem must remember the room left, $\text{OPT}(i, w)$;
   traceback by comparing with the entry above.

3. **Pseudo-polynomial:** $O(nW)$ is polynomial in the value of W,
   but exponential in its number of bits.

> LCS in $O(nm)$ time.
> Subset sum in $O(nW)$ time.

Reference: CLRS, Chapters 14.3, 14.4; KT, Chapter 6.4