Dynamic programming

More examples

A1

: 10×100 (p

0

Example 1: Longest common subsequence

Alphabet = set of characters:

{a,b,c,. . . ,z}, {A,C,G,T}, {0,1}

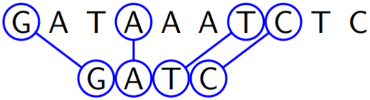
String = sequence $x_1x_2x_3 \ldots x_n$ of characters from an alphabet:

hello world

GATAAATCTGGTCTTATTTCC

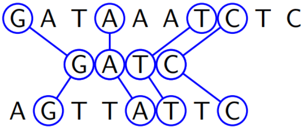
0010110010101010001111

Subsequence = subset of the characters in the string, in unchanged order:



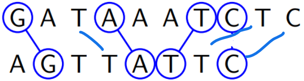
Longest common subsequence

Common subsequence for two strings:



Or simply:

deleting C -C
to make room for T- T



a even longer subsequence could be, by making new connections and push the connections around

Longest common subsequence (LCS):

$$X = x_1 x_2 x_3 \ldots x_m$$
$$Y = y_1 y_2 y_3 \ldots y_n$$

For two strings of length m and n, find their longest common subsequence.

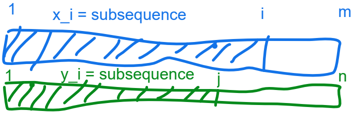
The length of this can be seen as a measure of similarity between the strings (e.g., DNA sequences).

The longer the common subsquence, the more the two DNA strings are similar

Recursive solution?

whole string = X

Whole string = Y



We want to work on creating a recursive solution. Therefore, we define a notion for subproblems:

- $X_i = x_1x_2x_3 \ldots x_i$ for $1 \leq i \leq m$
- $Y_j = y_1y_2y_3 \ldots$ y□ for $1 \leq j \leq n$
- X_0 and Y_0 are the empty string   if i or j = 0, it is an empty string
- **lcs**(i, j) is the length of the longest common subsequence of X_i and Y_j

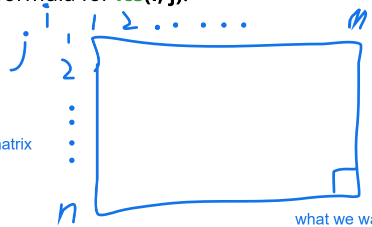
We want to find **lcs(m, n)**.

More generally: We are looking for a recursive formula for **lcs(i, j)**.

Base case: Clearly, **lcs**(0, j) = **lcs**(i, 0) = 0.

ofcause is one is zero/empty it canm not
have anything i common with another string

can be written as a i,j matrix



what we want to find, the longestt common subseq

Optimal subproblems I

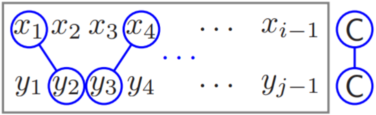
Formula for lcs(i ,j): Case I: $x_i = y_j$

Observation: A common subsequence Z for Xi and Yj of length k consists of:

- A last character is **zk**.
- A string **Z'** = $z_1\ z_2\ z_3\ \ldots\ z_{k-1}$ of length *k - 1*, which must be a common subsequence of **X_{i-1}** and **Y_{j-1}** (the characters in **Z** must appear in the same order as in **X** and **Y**, so only the last character in **Z** can possibly be **xi** and **yj**).

Observation (optimal subproblems) for Case I:

If Z is a longest common subsequence for Xi and Yj, then Z' must be a longest common subsequence of X_{i-1} and Y_{j-1}. For if there were a longer common subsequence for X_{i-1} and Y_{j-1}, it could be extended by the character xi (= yj) and become a longer common subsequence for Xi and Yj.



A1
: 10×100 (p
0

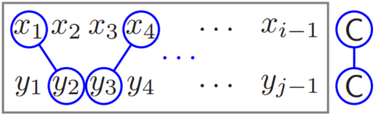
Optimal subproblems I

on the last placement i and j in the
two strings, the element is the same

From the observation in Case I (xi = yj):

length common subsequence

- **lcs**(i, j) = **lcs**(i − 1, j − 1) + 1
- A longest common subsequence for Xi−1 and Yj−1, when extended by the character xi (= yj), is a longest common subsequence for Xi and Yj.



Optimal subproblems II

the last element from x and from y is not the same

Formula for lcs(i ,j): Case II: $x_i \neq y_j$

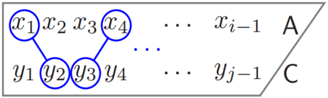
Observation: A common subsequence Z = $z_1z_2z_3$ ... zk for Xi and Yj cannot have zk as a match of xi and yj (since these are different).

Thus, Z must be a common subsequence for either Xi−1 and Yj, or for Xi and Yj−1 (or possibly both).

Observation (optimal subproblems) for Case II:

> If Z is a longest common subsequence for Xi and Yj, then it must be a longest common subsequence for either Xi−1 and Yj, or for Xi and Yj−1 (or possibly both). For if there were a longer common subsequence for either Xi−1 and Yj, or for Xi and Yj−1, this would also be a longer common subsequence for Xi and Yj.

-1 here



-1 here

Optimal subproblems II

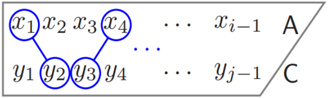
Let T1 be a longest common subsequence for Xi−1 and Yj, and let T2 be a longest common subsequence for Xi and Yj−1.

From the observation in Case II (xi ≠ yj), it follows that, among T1 and T2, at least one of them is a longest common subsequence for Xi and Yj.

Neither T1 nor T2 can be longer than the longest common subsequence for Xi and Yj (since both are subsequences of Xi and Yj).

Thus, from the observation in Case II (xi ≠ yj):

- lcs(i, j) = max(lcs(i − 1, j), lcs(i, j − 1))
- If lcs(i − 1, j) ≥ lcs(i, j − 1), then a longest common subsequence for Xi−1 and Yj is also a longest common subsequence for Xi and Yj. A symmetric statement holds for "≤" and Xi and Yj−1.



Recursive formula for lcs(i,j)

All in all, we have found the following recursive formula for lcs(i, j):

$$\mathrm{lcs}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \mathrm{lcs}(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(\mathrm{lcs}(i-1, j), \mathrm{lcs}(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

It leads to a natural, simple recursive algorithm.

BUT: It is easy to see that there are repetitions among the subproblems' subproblems.

Thus, the same subproblems are repeatedly computed in different places in the recursion tree, and the runtime becomes very poor.

This can potentially be solved with memoization: have a table with space for the answer to all possible subproblems lcs(i, j), and store the answer when it is computed for the first time. Later, simply look it up.

Dynamic programming: Instead of using recursion, directly fill in this table bottom-up in a structured manner.

Dynamic programming

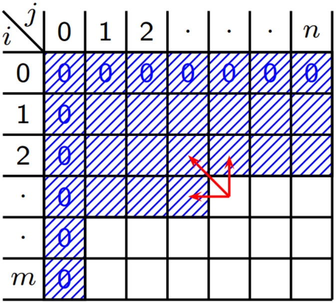
Dynamic programming: Fill in the table for lcs(i, j) bottom-up in a structured manner.

$i \backslash j$	0	1	2	·	·	·	n
0	0	0	0	0	0	0	0
1	0						
2	0						
·	0						
·	0						
m	0						

$$\mathrm{lcs}(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \mathrm{lcs}(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(\mathrm{lcs}(i-1, j), \mathrm{lcs}(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Dynamic programming

Dynamic programming: Fill in the table for lcs(i, j) bottom-up in a structured manner.

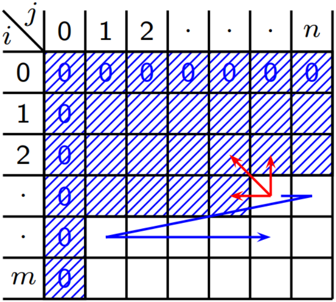


to fill in the next entry, would need to know the diagonal for when x_i -1 and y_i-1. And the perpendicular, for when only one of x_i and y_i is -1.

$$
\mathrm{lcs}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \mathrm{lcs}(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(\mathrm{lcs}(i-1, j), \mathrm{lcs}(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}
$$

Dynamic programming

Dynamic programming: fill in the table above lcs(i,j) bottom-up in a structured manner.

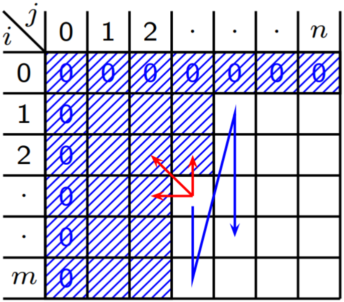


we need a structured why of
filling the matrix out.
fx from left to right, row for row

$$\mathrm{lcs}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \mathrm{lcs}(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(\mathrm{lcs}(i-1, j), \mathrm{lcs}(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Dynamic programming

Dynamic programming: fill in the table above lcs(i,j) bottom-up in a structured manner.



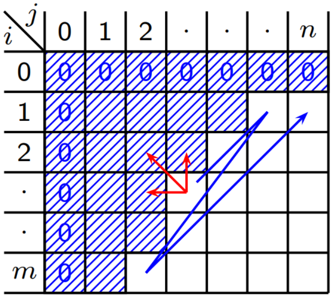
this is also a legit structured way of filling out the matrix, from top to botttom colum for coulm

$$\text{lcs}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \text{lcs}(i-1, j-1) + 1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ \max(\text{lcs}(i-1,j), \text{lcs}(i,j-1)) & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases}$$

A1 10×100 (p
0
8 / .æ/ . . &n
.; / .*n

Dynamic programming

Dynamic programming: fill in the table above lcs(i,j) bottom-up in a structured manner.



also legit, going diagonal

every method can be atrived
with just two for loops

$$\mathsf{lcs}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \mathsf{lcs}(i-1, j-1) + 1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ \max(\mathsf{lcs}(i-1,j), \mathsf{lcs}(i,j-1)) & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Running time

Dynamic programming: fill in the table above lcs(i,j) bottom-up in a structured manner.

$i \backslash j$	0	1	2	$\cdot$	$\cdot$	$\cdot$	n
0	0	0	0	0	0	0	0
1	0						
2	0						
$\cdot$	0						
$\cdot$	0						
m	0						

Table size: mn

Fill table entry: O(max size of the red graph) = O(1).

Total time: O(the product of the two) = O(mn).

Find a concrete solution

lcs(m, n) is the length of a longest common subsequence for X = Xm and Y = Yn. If we want to find a specific common subsequence of this length:

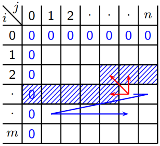
For each cell in the table, store which of the three red arrows gave the lcs(i, j) value for that cell.

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	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

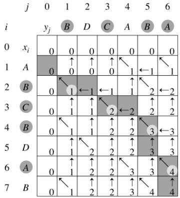
Follow the stored arrows backward from lcs(m, n). When a diagonal arrow is followed, it is Case I, and xi (= yj) is printed. Otherwise, it is Case II, and nothing is printed. In total, a longest common subsequence for X and Y is printed in reverse order in O(m + n) time.

Space usage for LCS

If we only need the length of the longest common subsequence, we can use min{m, n} space.



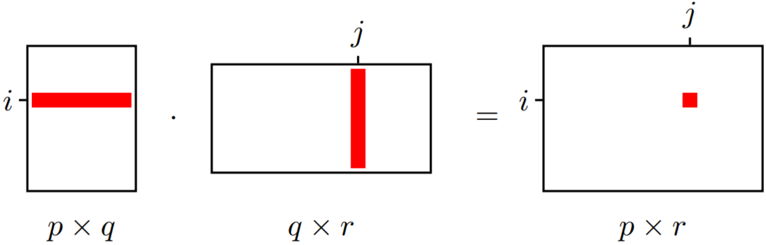
If we need the actual longest common subsequence, we must store the entire table, i.e., use Θ(mn) space (since we don't know the path back, we must store the entire table).



[Hirschberg proposed a method in 1975 to achieve this with min{m, n} space]

Example 2: Multi-Matrix Multiplication

A p × q matrix A1 and a q × r matrix A2 can be multiplied in O(pqr) time.
The result is a **p × r** matrix.



Matrix multiplication is associative:

$$A_1 \cdot (A_2 \cdot A_3) = (A_1 \cdot A_2) \cdot A_3$$

Multi-Matrix Multiplication

Matrix multiplication is associative:

$$A_1 \cdot (A_2 \cdot A_3) = (A_1 \cdot A_2) \cdot A_3$$

But the running time is NOT the same. Example:

A_1	A_2	A_3
10×100	100×5	5×50

$(A_2 \cdot A_3)$:		100×50
$(A_1 \cdot A_2)$:	10×5	

Time for $A_1 \cdot (A_2 \cdot A_3)$ is 10.100.50 +100.5.50 =75.000

Time for $(A_1 \cdot A_2) \cdot A_3$ is 10.100.5 +10.5.50 =7.500

Multi-Matrix Multiplication

The question:

For a product of n matrices:

$$A_1.A_2.A_3\cdots A_{n-1}.A_n$$

with compatible dimensions:

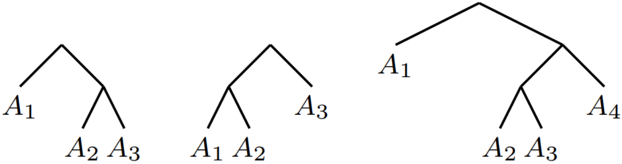
$$\begin{array}{cccccc} A_1 & A_2 & A_3 & \ldots & A_{n-1} & A_n \\ p_0 \times p_1 & p_1 \times p_2 & p_2 \times p_3 & \ldots & p_{n-2} \times p_{n-1} & p_{n-1} \times p_n \end{array}$$

the dimentions of the matrix
match like it should in matrix multiplication

What is the minimum time to multiply them together?

Computational trees

Order = parenthesis notation = binary computation tree:



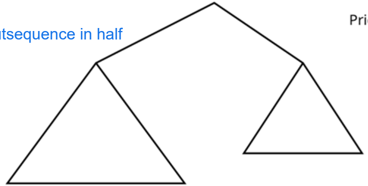
$A_1(A_2A_3)$ $(A_1A_2)A_3$ $A_1((A_2A_3)A_4)$

Optimal subproblems and recursive equation

Let m(i, j) be the cost of the best way to multiply Ai, ..., Aj together.

Observation (optimal subproblems):

The subtrees of the root of an optimal tree must themselves be optimal computation trees.



Try all root locations, i.e. all splits Ai , . . . ,Ak and Ak+1, . . . ,Aj:

$$m(i,j) = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{m(i,k) + m(k+1,j) + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

base case

m(i,j) =minimum cost need to multiply matrices from Ai to Aj together

left subtree+ right subtree+ root

A
1

8 / .æ/ . . &n

Table

Repetitions among subproblems' subproblems.Create a table and fill it systematically. The goal is to find m(1, n).

$$m(i,j) = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{m(i,k) + m(k+1,j) + p_{i-1}p_k p_j\} & \text{if } i < j \end{cases}$$

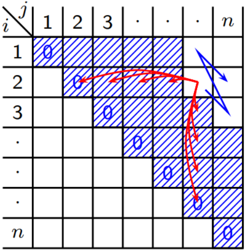


Table size: O(n^2).

Fill table entry: O(max size of the red graph) = O(n).

Total time: O(product of the two) = O(n^3).

Find a concrete solution: follow the optimal choices backward.