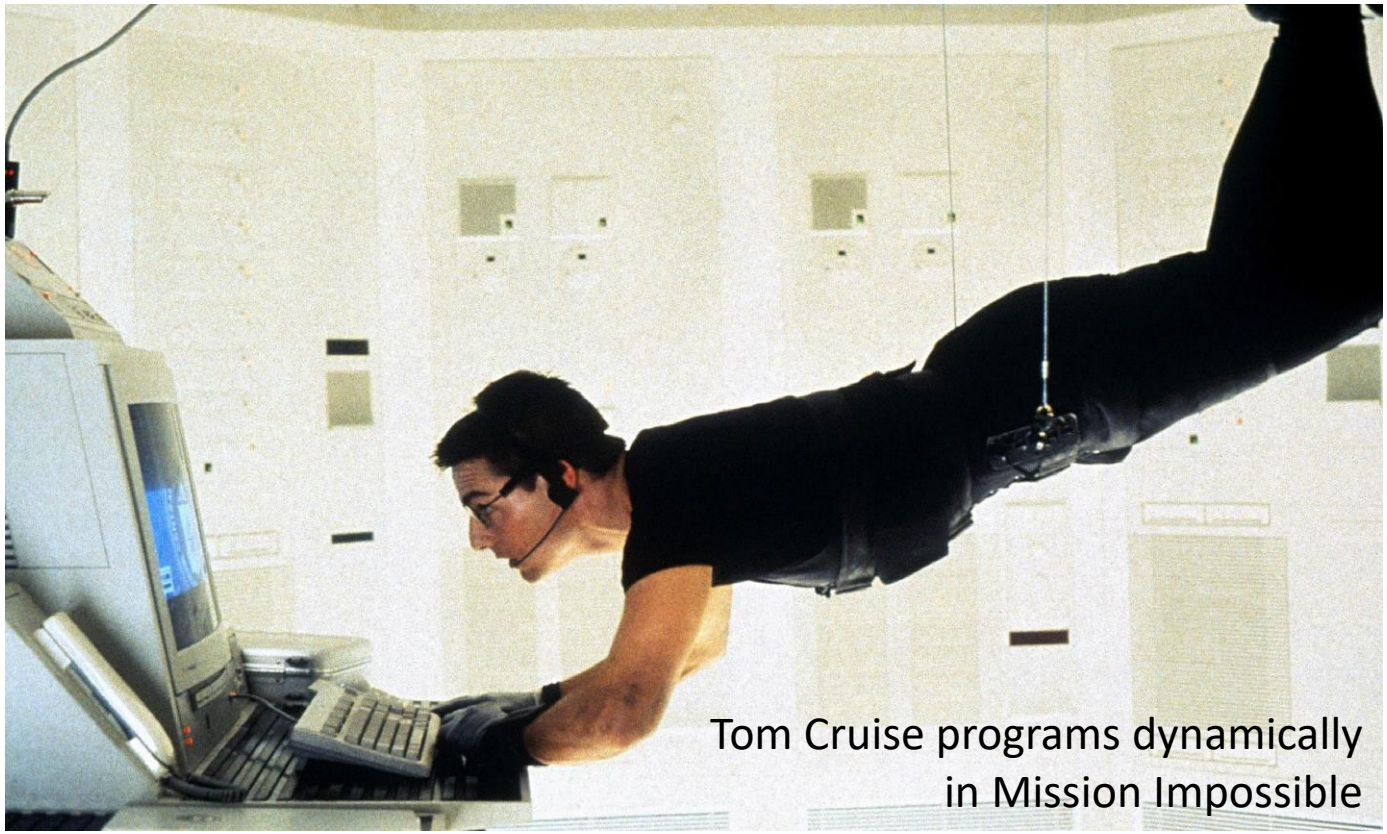
Advanced Data Structures and Algorithms

More dynamic programming!

Longest Common Subsequences

Last time

Dynamic Programming!



Tom Cruise programs dynamically in Mission Impossible

Last time

Dynamic Programming!

- Not coding in an action movie.

Last time

Dynamic Programming!

- Dynamic programming is an algorithm design paradigm.
- Basic idea:
  - Identify **optimal sub-structure**
    - Optimum to the big problem is built out of optima of small sub-problems
  - Take advantage of **overlapping sub-problems**
    - Only solve each sub-problem once, then use it again and again
  - Keep track of the solutions to sub-problems in a table as you build to the final solution.

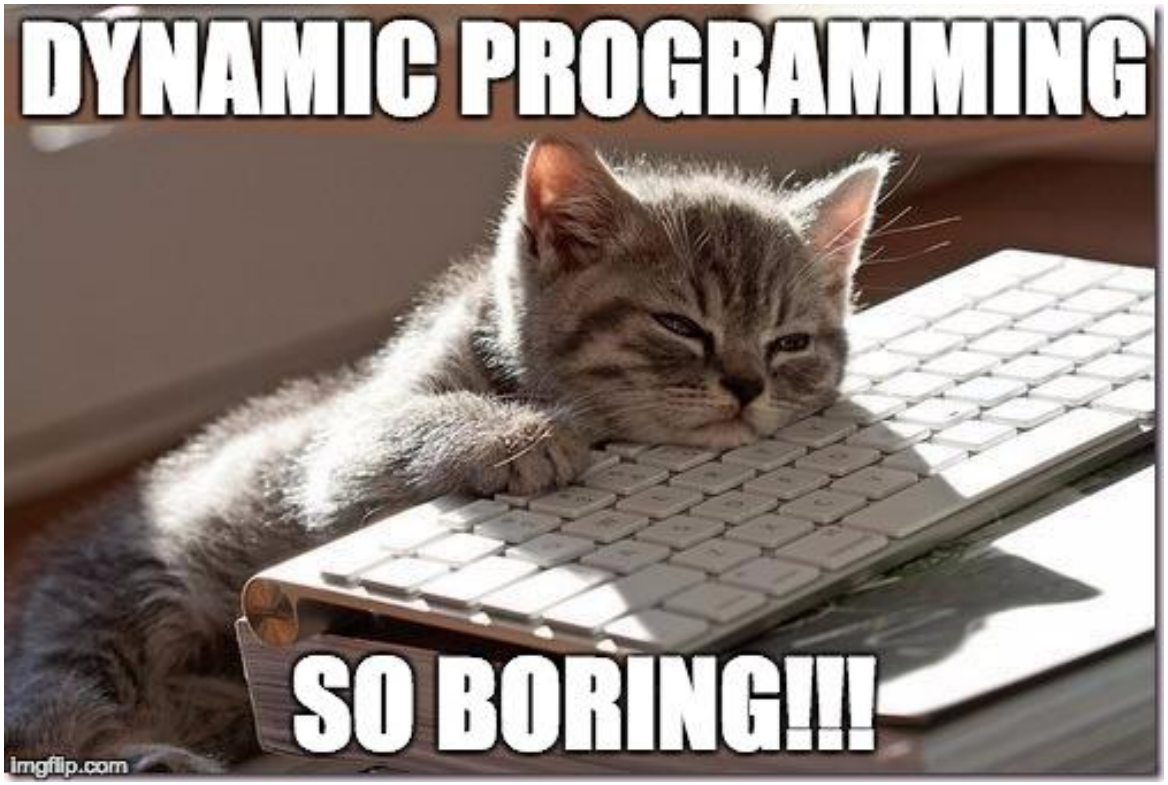
Rest of Dynamic Programming

- Examples of dynamic programming:
  1. Longest common subsequence

The goal of this lecture

The goal of this lecture

- For you to get really bored of dynamic programming



- How similar are these two species?

Longest Common Subsequence

- How similar are these two species?

DNA:

AGCCCTAAGGGCTACCTAGCTT

DNA:

GACAGCCTACAAGCGTTAGCTTG

Longest Common Subsequence

- How similar are these two species?

DNA:

AGCCCTAAGGGCTACCTAGCTT

DNA:

GACAGCCTACAAGCGTTAGCTTG

- Pretty similar, their DNA has a long common subsequence:

AGCCTAAGCTTAGCTT

Longest Common Subsequence

- Subsequence:
  - BDFH is a **subsequence** of ABCDEFGH

Longest Common Subsequence

- Subsequence:
  - BDFH is a **subsequence** of ABCDEFGH
- If X and Y are sequences, a **common subsequence** is a sequence which is a subsequence of both.
  - BDFH is a **common subsequence** of ABCDEFGH and of ABDFGHI

Longest Common Subsequence

- Subsequence:
  - BDFH is a **subsequence** of ABCDEFGH
- If X and Y are sequences, a **common subsequence** is a sequence which is a subsequence of both.
  - BDFH is a **common subsequence** of ABCDEFGH and of ABDFGHI
- A **longest common subsequence**…
  - …is a common subsequence that is longest.
  - The **longest common subsequence** of ABCDEFGH and ABDFGHI is ABDFGH.

We sometimes want to find these

- Applications in bioinformatics

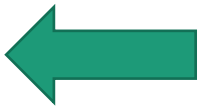
- The unix command diff

```
DN0a22a660:~ mary$ cat file1
A
B
C
D
E
F
G
H
DN0a22a660:~ mary$ cat file2
A
B
D
F
G
H
I
DN0a22a660:~ mary$ diff file1 file2
3d2
< C
5d3
< E
8a7
> I
DN0a22a660:~ mary$
```

Recipe for applying Dynamic Programming

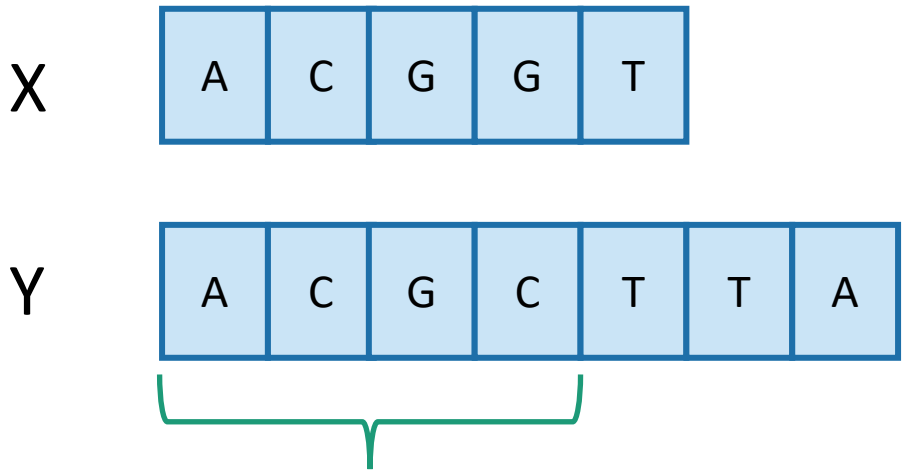
- **Step 1:** Identify optimal substructure.
- **Step 2:** Find a recursive formulation for the length of the longest common subsequence.
- **Step 3:** Use dynamic programming to find the length of the longest common subsequence.
- **Step 4:** If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual LCS.
- **Step 5:** If needed, code this up like a reasonable person.

Recipe for applying Dynamic Programming

- **Step 1:** Identify optimal substructure. 
- **Step 2:** Find a recursive formulation for the length of the longest common subsequence.
- **Step 3:** Use dynamic programming to find the length of the longest common subsequence.
- **Step 4:** If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual LCS.
- **Step 5:** If needed, code this up like a reasonable person.

Step 1: Optimal substructure

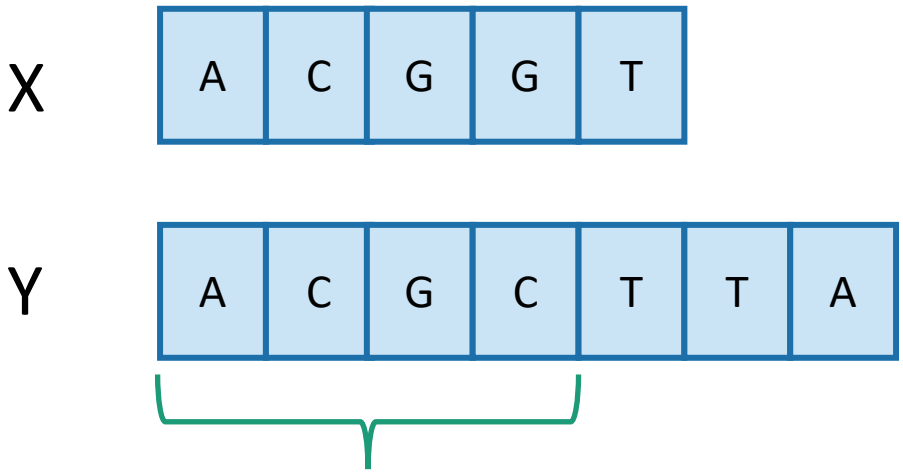
Prefixes:



Notation: denote this prefix **ACGC** by Y_4

Step 1: Optimal substructure

Prefixes:



Notation: denote this prefix **ACGC** by Y_4

- Our sub-problems will be finding LCS's of prefixes to X and Y.
- Let C[i,j] = length_of_LCS(X_i, Y_j)

Examples: C[2,3] = 2
C[4,4] = 3

Optimal substructure ctd.

- Subproblem:
  - finding LCS's of prefixes of X and Y.

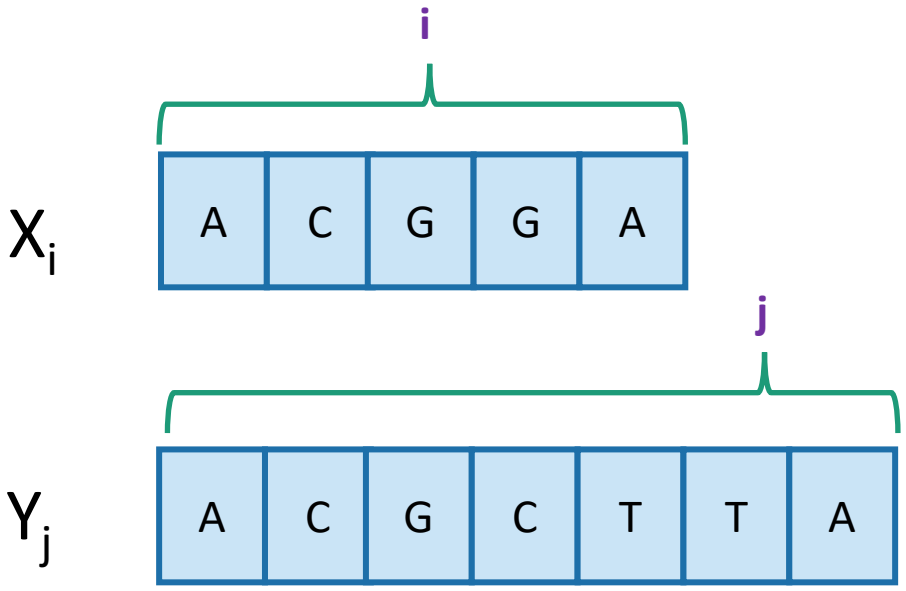
- Why is this a good choice?
  - As we will see, there's some relationship between LCS's of prefixes and LCS's of the whole things.
  - These subproblems overlap a lot.

Recipe for applying Dynamic Programming

- **Step 1:** Identify optimal substructure.
- **Step 2:** Find a recursive formulation for the length of the longest common subsequence.
- **Step 3:** Use dynamic programming to find the length of the longest common subsequence.
- **Step 4:** If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual LCS.
- **Step 5:** If needed, code this up like a reasonable person.

Goal

- Write C[i,j] in terms of the solutions to smaller sub-problems

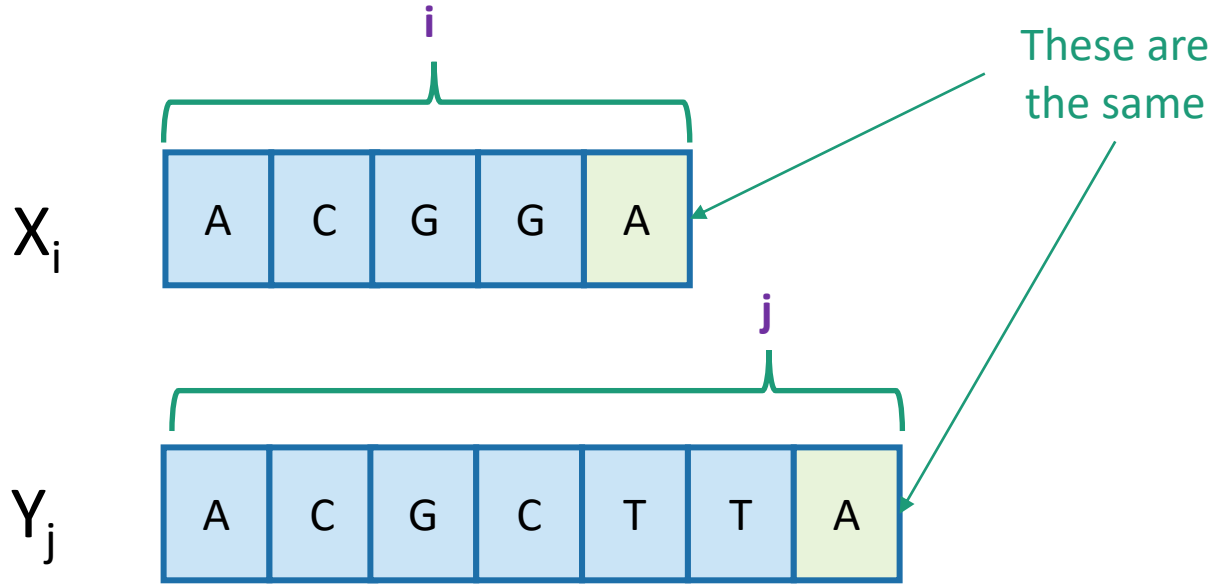


C[i,j] = length_of_LCS(X_i, Y_j)

Two cases

- Our sub-problems will be finding LCS's of prefixes to X and Y.
- Let C[i,j] = length_of_LCS(X_i, Y_j)

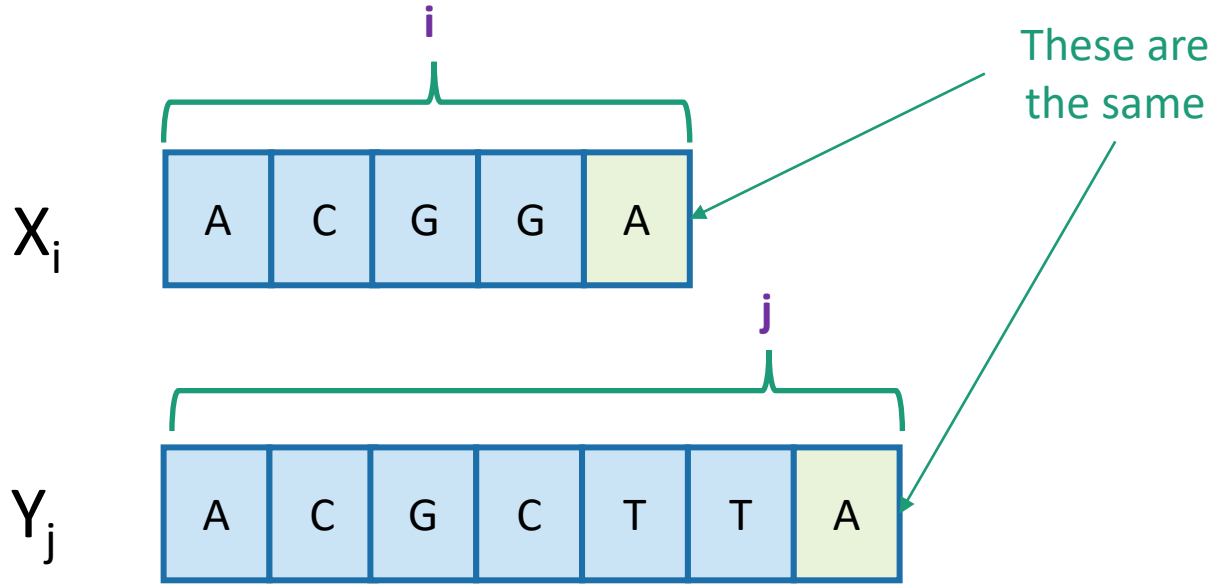
Case 1: X[i] = Y[j]



Two cases

- Our sub-problems will be finding LCS's of prefixes to X and Y.
- Let C[i,j] = length_of_LCS(X_i, Y_j)

Case 1: X[i] = Y[j]

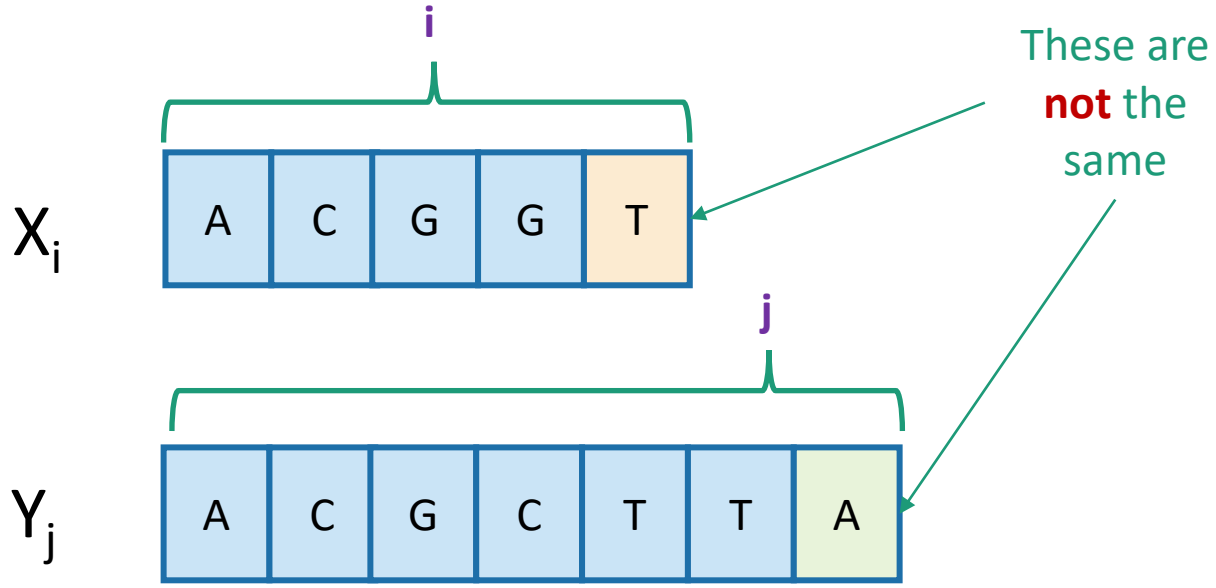


- Then C[i,j] = 1 + C[i-1,j-1].
  - because LCS(X_i,Y_j) = LCS(X_{i-1},Y_{j-1}) followed by A

Two cases

- Our sub-problems will be finding LCS's of prefixes to X and Y.
- Let C[i,j] = length_of_LCS(X_i, Y_j)

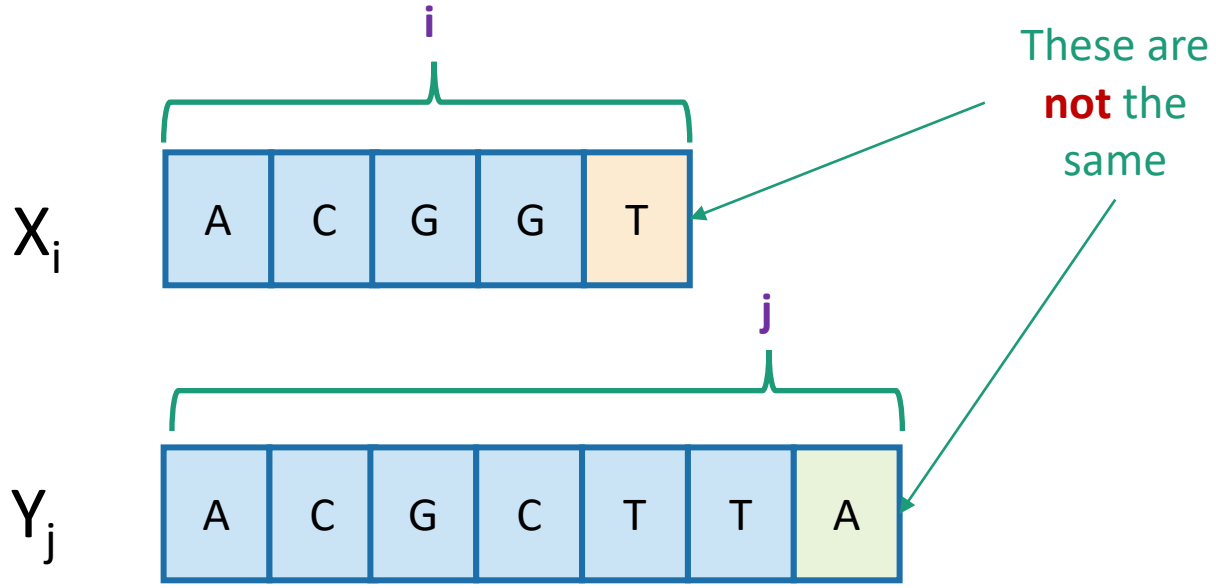
Case 2: X[i] != Y[j]



Two cases

- Our sub-problems will be finding LCS's of prefixes to X and Y.
- Let C[i,j] = length_of_LCS(X_i, Y_j)

Case 2: X[i] != Y[j]

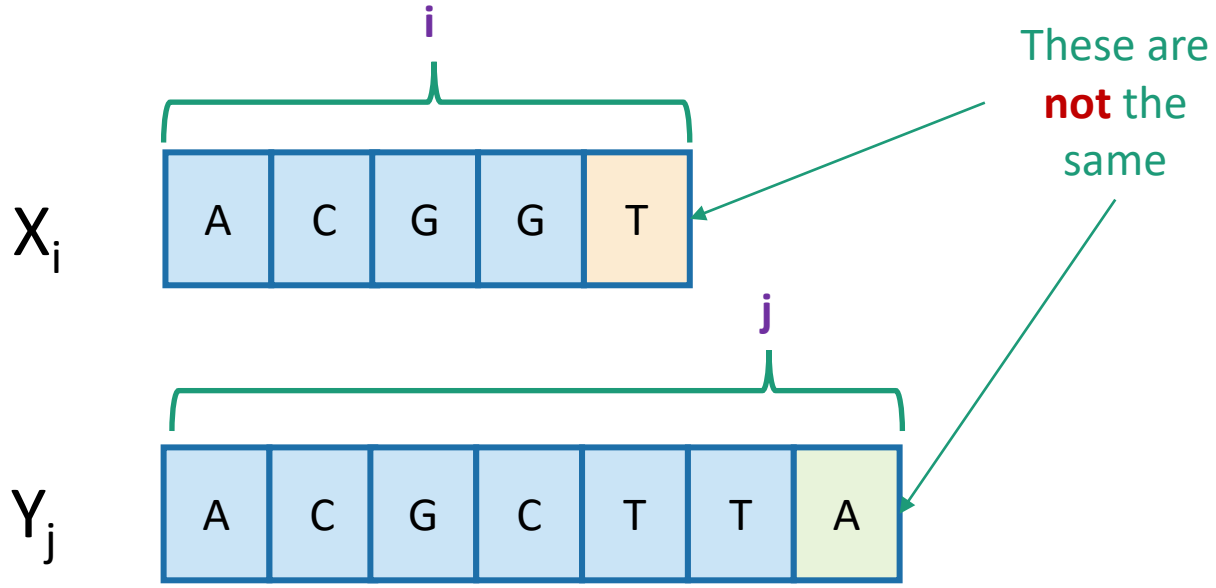


- Then C[i,j] = max{ C[i-1,j], C[i,j-1] }.

Two cases

- Our sub-problems will be finding LCS's of prefixes to X and Y.
- Let C[i,j] = length_of_LCS(X_i, Y_j)

Case 2: X[i] != Y[j]

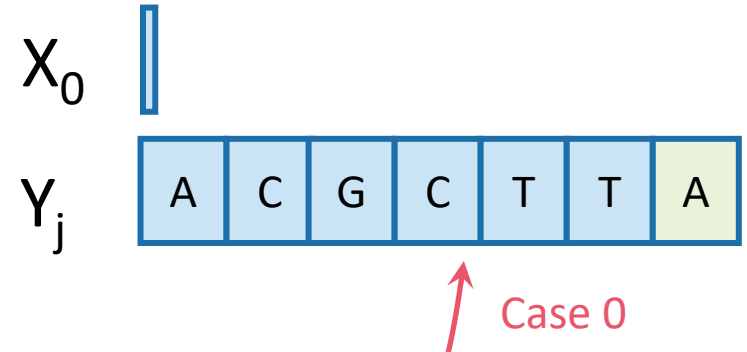


- Then C[i,j] = max{ C[i-1,j], C[i,j-1] }.
  - either $LCS(X_i,Y_j) = LCS(X_{i-1},Y_j)$ and T is not involved,
  - or $LCS(X_i,Y_j) = LCS(X_i,Y_{j-1})$ and A is not involved,
  - (maybe both are not involved, that's covered by the "or").

Recursive formulation
of the optimal solution

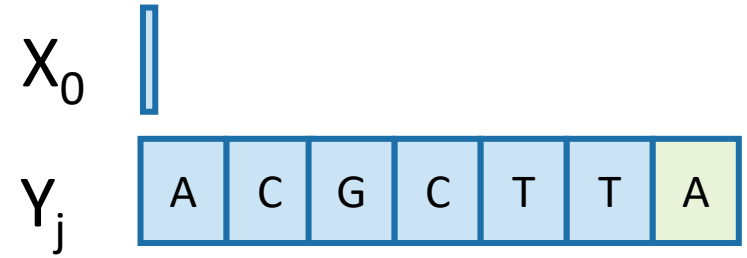
- $C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$

Recursive formulation
of the optimal solution



- $C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$

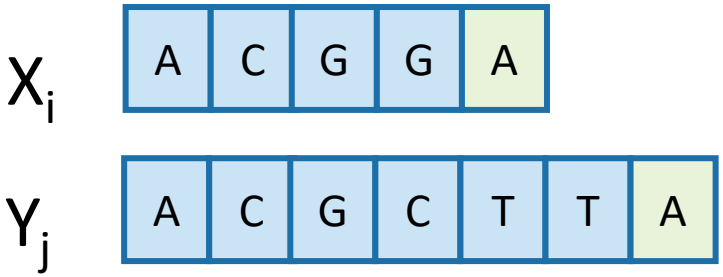
Recursive formulation
of the optimal solution



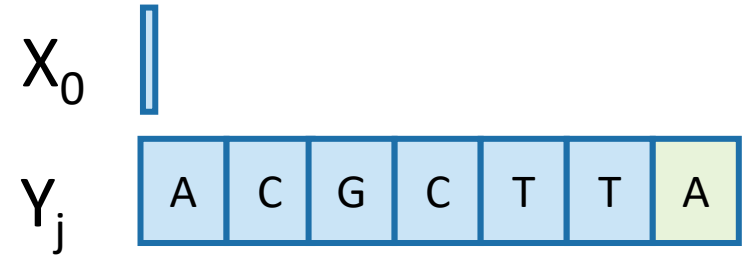
Case 0

- $C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$

Case 1



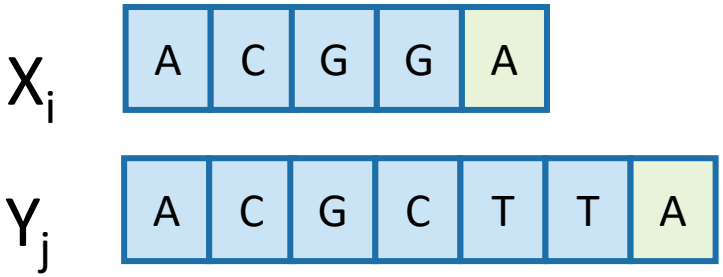
Recursive formulation
of the optimal solution



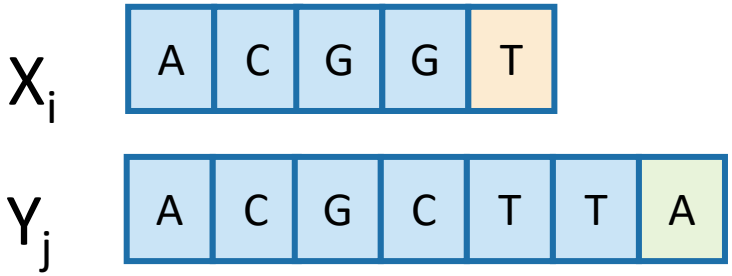
Case 0

- $C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$

Case 1



Case 2



Recipe for applying Dynamic Programming

- **Step 1:** Identify optimal substructure.
- **Step 2:** Find a recursive formulation for the length of the longest common subsequence.
- **Step 3:** Use dynamic programming to find the length of the longest common subsequence.
- **Step 4:** If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual LCS.
- **Step 5:** If needed, code this up like a reasonable person.

LCS DP

- **LCS**(X, Y):
  - C[i,0] = C[0,j] = 0 for all i = 0,…,m, j=0,…n.
  - **For** i = 1,…,m and j = 1,…,n:
    - **If** X[i] = Y[j]:
      - C[i,j] = C[i-1,j-1] + 1
    - **Else:**
      - C[i,j] = max{ C[i,j-1], C[i-1,j] }
  - Return C[m,n]

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

LCS DP

- **LCS**(X, Y):
  - C[i,0] = C[0,j] = 0 for all i = 0,…,m, j=0,…n.
  - **For** i = 1,…,m and j = 1,…,n:
    - **If** X[i] = Y[j]:
      - C[i,j] = C[i-1,j-1] + 1
    - **Else:**
      - C[i,j] = max{ C[i,j-1], C[i-1,j] }
  - Return C[m,n]

Running time: O(nm)

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A				
C				
G				
G				
A				

0	0	0	0	0
0				
0				
0				
0				
0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A				
C				
G				
G				
A				

0	0	0	0	0
0	1			
0				
0				
0				
0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A	0	0	0	0	0
C	0	1	1		
G	0				
G	0				
A	0				
	0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A				
C				
G				
G				
A				

0	0	0	0	0
0	1	1	1	
0				
0				
0				
0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

	A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0				
G	0				
G	0				
A	0				

X

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A	0	0	0	0	0
C	0	1	1	1	1
G	0	1			
G	0				
A	0				
	0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X	A	C	G	G	A

Y	A	C	T	G

		Y			
		A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2		
G	0				
G	0				
A	0				

X

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X	A	C	G	G	A

Y	A	C	T	G

		Y			
		A	C	T	G
X		0	0	0	0	0
A		0	1	1	1	1
C		0	1	2	2	
G		0				
G		0				
A		0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

	A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2	2	2
G	0				
G	0				
A	0				

X

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X		0	0	0	0	0
A		0	1	1	1	1
C		0	1	2	2	2
G		0	1	2	2	3
G		0				
A		0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X		0	0	0	0	0
A		0	1	1	1	1
C		0	1	2	2	2
G		0	1	2	2	3
G		0	1	2	2	3
A		0				

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X	A	C	G	G	A

Y	A	C	T	G

		Y			
		A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2	2	2
G	0	1	2	2	3
G	0	1	2	2	3
A	0	1	2	2	3

X

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

		Y			
		A	C	T	G
X	A	0	0	0	0	0
	C	0	1	1	1	1
	G	0	1	2	2	2
	G	0	1	2	2	3
	A	0	1	2	2	3

So the LCM of X and Y has length 3.

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Recipe for applying Dynamic Programming

- **Step 1:** Identify optimal substructure.
- **Step 2:** Find a recursive formulation for the length of the longest common subsequence.
- **Step 3:** Use dynamic programming to find the length of the longest common subsequence.
- **Step 4:** If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual LCS.
- **Step 5:** If needed, code this up like a reasonable person.

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

	A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2	2	2
G	0	1	2	2	3
G	0	1	2	2	3
A	0	1	2	2	3

X

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X	A	C	G	G	A

Y	A	C	T	G

X \ Y		A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2	2	2
G	0	1	2	2	3
G	0	1	2	2	3
A	0	1	2	2	3

- Once we've filled this in, we can work backwards.

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A
C
G
G
A

0	0	0	0	0
0	1	1	1	1
0	1	2	2	2
0	1	2	2	3
0	1	2	2	3
0	1	2	2	3

- Once we've filled this in, we can work backwards.

That 3 must have come from the 3 above it.

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

	A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2	2	2
G	0	1	2	2	3
G	0	1	2	2	3
A	0	1	2	2	3

X

- Once we've filled this in, we can work backwards.
- A diagonal jump means that we found an element of the LCS!

This 3 came from that 2 – we found a match!

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A	0	0	0	0	0
C	0	1	1	1	1
G	0	1	2	2	2
G	0	1	2	2	3
A	0	1	2	2	3
	0	1	2	2	3

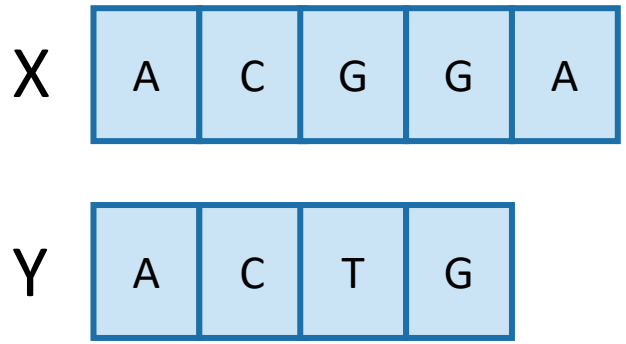
- Once we've filled this in, we can work backwards.
- A diagonal jump means that we found an element of the LCS!

That 2 may as well have come from this other 2.

G

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example



Y

	A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2	2	2
G	0	1	2	2	3
G	0	1	2	2	3
A	0	1	2	2	3

X

- Once we've filled this in, we can work backwards.
- A diagonal jump means that we found an element of the LCS!

G

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{ C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

	A	C	T	G
	0	0	0	0	0
A	0	1	1	1	1
C	0	1	2	2	2
G	0	1	2	2	3
G	0	1	2	2	3
A	0	1	2	2	3

X

- Once we've filled this in, we can work backwards.
- A diagonal jump means that we found an element of the LCS!

C	G

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Example

X

A	C	G	G	A

Y

A	C	T	G

Y

A	C	T	G

X

A	0	0	0	0	0
C	0	1	1	1	1
G	0	1	2	2	2
G	0	1	2	2	3
A	0	1	2	2	3
	0	1	2	2	3

- Once we've filled this in, we can work backwards.
- A diagonal jump means that we found an element of the LCS!

A	C	G

This is the LCS!

$$C[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \text{ and } i, j > 0 \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } X[i] \neq Y[j] \text{ and } i, j > 0 \end{cases}$$

Finding an LCS

- Takes time O(mn) to fill the table
- Takes time O(n + m) on top of that to recover the LCS
  - We walk up and left in an n-by-m array
  - We can only do that for n + m steps.
- Altogether, we can find LCS(X,Y) in time O(mn).

Recipe for applying Dynamic Programming

- **Step 1:** Identify optimal substructure.
- **Step 2:** Find a recursive formulation for the length of the longest common subsequence.
- **Step 3:** Use dynamic programming to find the length of the longest common subsequence.
- **Step 4:** If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual LCS.
- **Step 5:** If needed, code this up like a reasonable person.

This pseudocode actually isn’t so bad

This pseudocode actually isn't so bad

- If we are only interested in the length of the LCS we can do a bit better on space:
  - Since we go across the table one-row-at-a-time, we can only keep two rows if we want.
- If we want to recover the LCS, we need to keep the whole table.

This pseudocode actually isn't so bad

- If we are only interested in the length of the LCS we can do a bit better on space:
  - Since we go across the table one-row-at-a-time, we can only keep two rows if we want.
- If we want to recover the LCS, we need to keep the whole table.

- Can we do better than O(mn) time?
  - A bit better.
    - By a log factor or so.

This pseudocode actually isn't so bad

- If we are only interested in the length of the LCS we can do a bit better on space:
  - Since we go across the table one-row-at-a-time, we can only keep two rows if we want.
- If we want to recover the LCS, we need to keep the whole table.

- Can we do better than O(mn) time?
  - A bit better.
    - By a log factor or so.
  - But doing much better (polynomially better) is an open problem!
    - If you can do it let me know !!!!!

What have we learned?

- We can find LCS(X,Y) in time O(nm)
  - if |Y|=n, |X|=m

- We went through the steps of coming up with a dynamic programming algorithm.
  - We kept a 2-dimensional table, breaking down the problem by decrementing the length of X and Y.

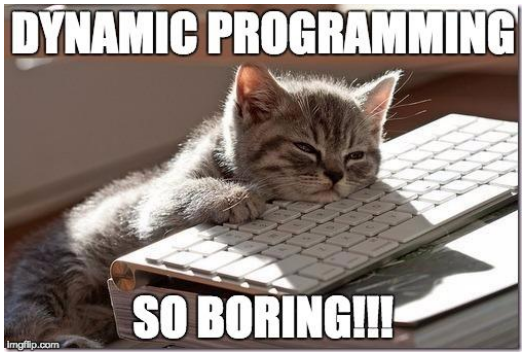
Recap

- We saw example of how to come up with dynamic programming algorithms.
  - Longest Common Subsequence
- There is a **recipe** for dynamic programming algorithms.

Recipe for applying Dynamic Programming

- **Step 1:** Identify optimal substructure.
- **Step 2:** Find a recursive formulation for the value of the optimal solution.
- **Step 3:** Use dynamic programming to find the value of the optimal solution.
- **Step 4:** If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual solution.
- **Step 5:** If needed, code this up like a reasonable person.

Recap



- We saw example of how to come up with dynamic programming algorithms.
  - Longest Common Subsequence

- There is a **recipe** for dynamic programming algorithms.
- Sometimes coming up with the right substructure takes some creativity

Acknowledgement

- Stanford University