5C Find a Longest Common Subsequence of Two Strings

Longest Common Subsequence Problem
Find a longest common subsequence of two strings.

Input: Two strings.
Output: A longest common subsequence of these strings.

Formatting

Input: Two newline-separated strings v and w.
Output: A longest common subsequence of v and w as a string.

Constraints

- The lengths of v and w will be between 1 and 10^3.
- Both v and w will be DNA strings.

Test Cases

Case 1

Description: The sample dataset is not actually run on your code.

Input:
```
GACT
ATG
```

Output:
```
AT
```

Case 2

Description: This dataset checks that you code solves the longest common *subsequence* problem instead of the related longest common *substring* problem. The longest common *substring* between **ACTG**AG and G**ACTG**G is ACGT, but the longest common *subsequence* between **ACGT**A**G** and G**ACTGG** is ACTGG.

Input:
```
ACTGAG
GACTGG
```

Output:
```
ACTGG
```

Case 3

Description: This simple dataset is used to check if your code correctly reconstructs the longest common subsequence from the backtracking matrix. Common errors include forgetting to reverse the reconstruction before returning (result: CA), terminating reconstruction too early (result: C), and starting reconstruction too late (result: A).

Input:
```
AC
AC
```

Output:
```
AC
```

Case 4

Description: This dataset checks that your code correctly considers the last character of each string. Off-by-one errors (can be caused by $0/1$ indexing errors) in indexing the input strings could result in the solution erroneously ignoring the final characters of inputs. If your code outputs an empty string it is likely that your implementation includes some off-by-one error.

Input:

```
GGGGT
CCCCT
```

Output:

```
T
```

Case 5

Description: This dataset checks that your code correctly considers the first character of each string. Off-by-one errors (can be caused by $0/1$ indexing errors) in indexing the input strings could result in the solution erroneously ignoring the first characters of inputs. If your code outputs an empty string it is likely that your implementation includes some off-by-one error.

Input:

```
TCCCC
TGGGG
```

Output:

```
T
```

Case 6

Description: This dataset checks that your code can handle inputs in which the two strings to be aligned are different lengths. If your output is incorrect make sure that your dynamic programming matrix has dimensions $(|v|+1)\times(|w|+1)$ or $(|w|+1)\times(|v|+1)$. If your code incorrectly sets the dynamic programming matrix dimensions to $(|v|+1)\times(|v|+1)$ or $(|w|+1)\times(|w|+1)$ it will not necessarily fail previous datasets since $|v|$ is the same as $|w|$ in all previous test datasets. Make sure that your implementation does not make any assumptions about the sizes of strings v and w.

Input:

```
AA
CGTGGAT
```

Output:

```
A
```

Case 7

Description: This dataset checks that your code can handle inputs in which the two strings to be aligned are different lengths. This dataset is similar to test dataset 6 except that in this dataset string v is longer than string w.

Input:

```
GGTGACGT
CT
```

Output:

```
CT
```

Case 8

Description: A larger dataset of the same size as that provided by the randomized autograder.