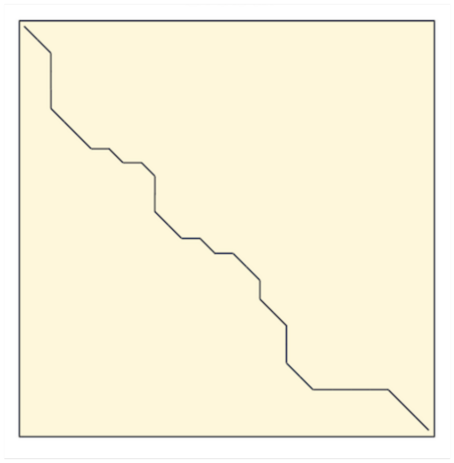
5E Find a Highest-Scoring Global Alignment of Two Strings

Global Alignment Problem
Find a highest-scoring global alignment between two strings using a scoring matrix.

Input: Two DNA strings, a match reward, a mismatch penalty, and an indel penalty.
Output: A highest-scoring global alignment of these two strings (with respect to the scoring parameters) and its score.



Formatting

Input: Three space-separated integers representing the match reward, mismatch penalty, and indel penalty respectively, followed by two newline-separated DNA strings v and w.
Output: The maximum global alignment score of v and w as an integer as determined by the scoring function followed by a newline-separated global alignment of v and w achieving this maximum score (if multiple global alignments achieving the maximum score exist, you may return any one).

Constraints

- The lengths of v and w will be between 1 and 10^4.
- Both v and w will be DNA strings.
- All parameters of the scoring function will be between 1 and 10^1.

Test Cases

Case 1

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

Input:
```
1 1 2
GAGA
GAT
```

Output:
```
-1
GAGA
GA-T
```

Case 2

Description: This test makes sure that your code correctly parses the first line of input and uses the correct penalties. The mismatch μ and indel σ penalties can easily be mistakenly swapped either when parsing the input or when actually applying the global alignment algorithm. In this case the mismatch penalty is more than twice the indel penalty. Therefore ending the alignment with two indels is better than simply aligning the mismatched final bases. If the mismatch and indel penalties were somehow switched in your code you will likely get a score of 1 and an alignment of `ACG` and `ACT`.

Input:
```
1 3 1
ACG
ACT
```

Output:
```
0
AC-G
ACT-
```

Case 3

Description: This test makes sure that the mismatch penalty is being correctly applied. A mismatch penalty of 1 means that an alignment making use of mismatched bases suffer a score decrease of 1. It can be easy to forget that penalties must be subtracted from the score, not added. Be sure that all penalties are being subtracted from score total when updating your dynamic programming matrix. Alternatively, you could negate the mismatch and indel penalties and add them to your scores. If your code outputs a score of 2 or 3 you are likely accidentally adding the penalties to your scores instead of subtracting them.

Input:
```
1 1 1
AT
AG
```

Output:
```
0
AT
AG
```

Case 4

Description: This test makes sure that your code allows for an output beginning with an indel. If your code doesn't make use of the base cases (first row and column of the dynamic programming matrix) scores then the correct score of 3 cannot be found. Be sure to correctly fill out your bases cases and consider them in your recursive cases.

Input:
```
2 5 1
TCA
CA
```

Output:
```
3
TCA
-CA
```

Case 5

Description: This test makes sure that your code can handle multiple indels in a row. If there is some indel specific error in reconstructing the alignment from your backtracking matrix you may be missing an indel at the beginning or end of the second output string. This test also makes sure that the correct score calculation is being used for global alignment. This particular dataset would have a score of 2 if fitting or local alignment were performed, but since this problem requires global alignment the preceding and trailing indels must be incorporated into the score.

Input:
```
1 10 1
TTTTCCTT
CC
```

Output:
```
-4
TTTTCCTT
----CC--
```

Case 6

Description: This test makes sure that your code can handles inputs in which the two strings differ drastically in length. Note that your reconstructed alignment may differ from the given output and is still correct as long as the "T" character in string w is aligned to one of the "T" characters in string v. If your output doesn't match the correct output make sure that your dynamic programming matrix has dimensions $(|v|+1)\times(|w|+1)$ or $(|w|+1)\times(|v|+1)$. If your code incorrectly set the dynamic programming matrix dimensions to $(|v|+1)\times(|v|+1)$ or $(|w|+1)\times(|w|+1)$ it will fail this case.

Input:
```
2 3 2
ACAGATTAG
T
```

Output:
```
-14
ACAGATTAG
-----T---
```

Case 7

Description: This test makes sure that your code can handles inputs in which the two strings differ drastically in length. This dataset is similar to test dataset 6 except in this dataset string v is much shorter than string w instead of vice-versa.

Input:

```
3 1 2
G
ACATACGATG
```

Output:

```
-15
------G----
ACATACGATAG
```

Case 8

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.