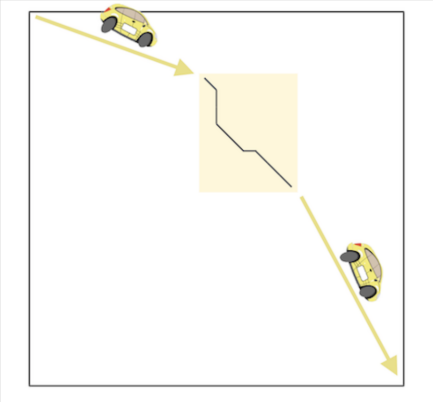
5F Find a Highest-Scoring Local Alignment of Two Strings

Local Alignment Problem
Find a highest-scoring local alignment between two strings.

Input: Two DNA strings, a match reward, a mismatch penalty, and an indel penalty.
Output: A highest-scoring local 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 local alignment score of v and w as an integer as determined by the scoring function followed by a newline-separated local alignment of v and w achieving this maximum score (if multiple local 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:

```
2
GA
GA
```

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 local alignment algorithm. If the mismatch and indel penalties were somehow switched in your code you will likely get a score of 5 and an alignment of `AGC` and `ATC`.

Input:

```
3 3 1
AGC
ATC
```

Output:

```
4
AG-C
A-TC
```

Case 3

Description: This test makes sure that the mismatch penalties are 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:

```
1
A
A
```

Case 4

Description: This test makes sure that your code can correctly find the highest scoring alignment, wherever it is in the dynamic programming matrix. This test also makes sure that your code correctly backtracks for local alignments. Be sure to terminate reconstruction of the aligned strings when a "free ride" can be used back to the origin. If your code doesn't correctly terminate reconstruction it is possible that your score will be correct but your alignment will be incorrect.

Input:

```
1 1 1
TAACG
ACGTG
```

Output:

```
3
ACG
ACG
```

Case 5

Description: This test makes sure that your code can handle inputs in which the strings vary drastically in length. If your output doesn't match the correct output make sure that your implementation doesn't make any assumptions about the lengths of the strings. 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 but it will fail this one.

Input:

```
3 2 1
CAGAGATGGCCG
ACG
```

Output:

```
6
CG
CG
```

Case 6

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 5 except that in this dataset string v is shorter than string w.

Input:

```
2 3 1
CTT
AGCATAAAGCATT
```

Output:

```
5
C-TT
CATT
```

Case 7

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