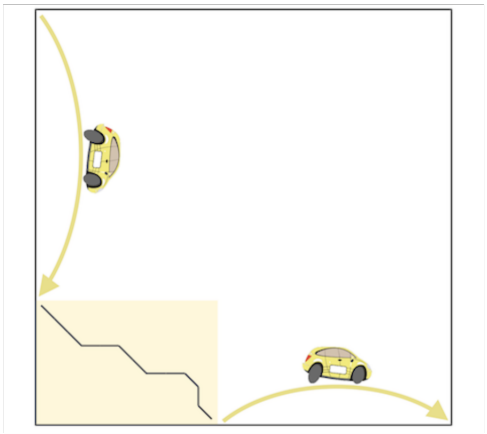
5I Find a Highest-Scoring Overlap Alignment of Two Strings

Overlap Alignment Problem
Find a highest-scoring overlap alignment between two strings.

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

Constraints

- The lengths of v and w will be between 1 and 10^3
- 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
GA
GA
```

Case 2

Description: This test makes sure that your dynamic programming matrix is correctly initialized. Skipping characters at the beginning of string v should not be associated with a score penalty since the suffix of string v is the only part of interest. In other words, we can prepend any number of gaps to string w without a score penalty. For example we can write this alignment as `CCAT` and `-AT` and simply ignore the gap sequence prepended to string w and the characters they align to in string v.

Input:

```
1 1 1
CCAT
AT
```

Output:

```
2
AT
AT
```

Case 3

Description: This test makes sure that your dynamic programming matrix is correctly penalizing indels in string v. Gaps at the beginning of a suffix of string v must be penalized. In this dataset the mismatch penalty is much higher than the indel penalty so that the ideal overlap alignment has a gap in the first character of the string v suffix. If your code outputs a score of 3 then it is likely that you're mistakenly not punishing gaps at the beginning of the suffix. If your code outputs an alignment similar to `AT` and `AT` then it's likely that your alignment reconstruction is incorrectly removing characters. While characters from the beginning of string v can be freely removed this is not the case for string w. Since our alignment uses the "C" character from string w we must also include the gap it aligns to in string v.

Input:

```
1 5 1
GAT
CAT
```

Output:

```
1
-AT
CAT
```

Case 4

Description: This test makes sure that your code correctly ignores characters at the end of string w if that results in a better alignment score. In overlap alignment only the prefix of string w must be aligned. Adding the "G" character to the alignment will only hurt the score, so it is not used in an ideal overlap alignment of this dataset. If your code includes the "G" character from string w in its output then make sure that you are selecting your final score from the correct place in your dynamic programming matrix. Also check to make sure your alignment reconstruction does not add extra characters to the final alignment.

Input:

```
1 5 1
ATCACT
AT
```

Output:

```
1
ACT
A-T
```

Case 5

Description: This test makes sure that your code correctly ignores characters at the end of string w if that results in a better alignment score. In overlap alignment only the prefix of string w must be aligned. Adding the "G" character to the alignment will only hurt the score, so it is not used in an ideal overlap alignment of this dataset. If your code includes the "G" character from string w in its output then make sure that you are selecting your final score from the correct place in your dynamic programming matrix. Also check to make sure your alignment reconstruction does not add extra characters to the final alignment.

Input:

```
1 1 5
ATCACT
ATG
```

Output:

```
0
CT
AT
```

Case 6

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 set the dynamic programming matrix dimensions to $(|v|+1)\times(|v|+1)$ or $(|w|+1)\times(|w|+1)$ it will fail this dataset.

Input:

```
3 2 1
CAGAGATGGCCG
ACG
```

Output:

```
5
-CG
ACG
```

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 shorter than string w.

Input:

```
2 3 1
CTT
AGCATAAAGCATT
```

Output:

```
0
--CT-T
AGC-AT
```

Case 8

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