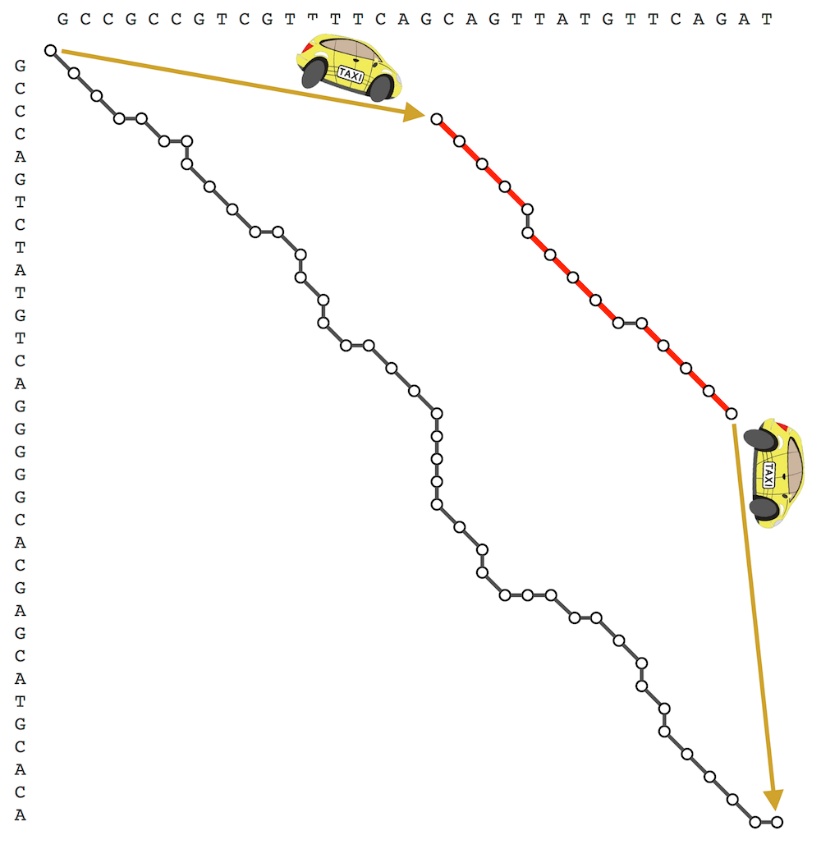
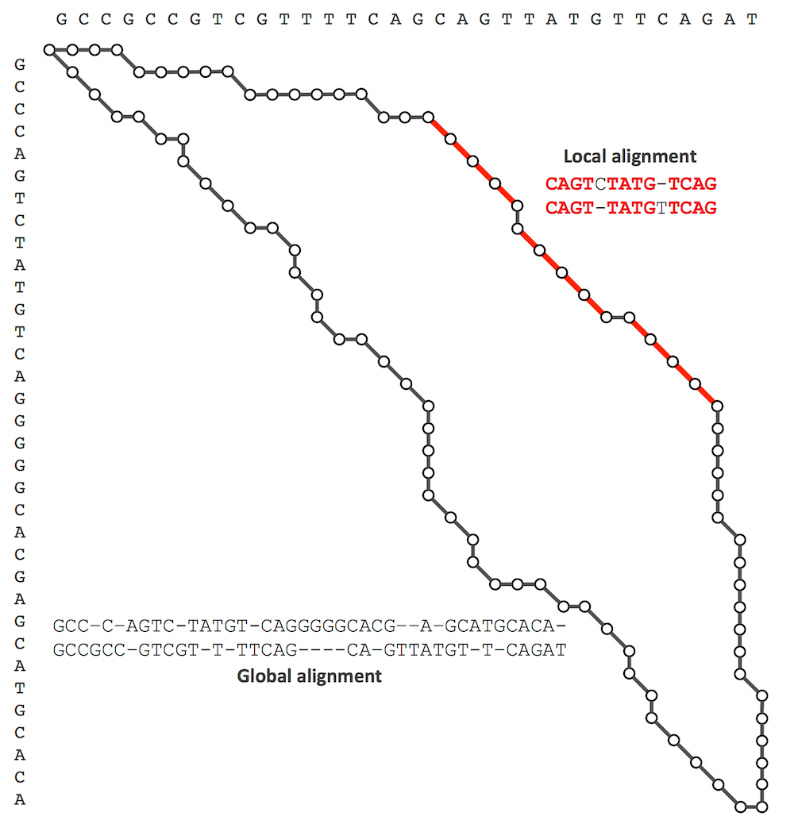
**Local Alignment Problem**

*Find a highest-scoring local alignment between two strings using a scoring matrix.*

**Input:** A match score *m*, a mismatch penalty *μ*, a gap penalty *σ*, and two DNA strings *s* and *t*.

**Output:** The maximum alignment score of a local alignment between *s* and *t* followed by a local alignment achieving this maximum score.

You are a Bioinformatician and you have obtained two gene sequences, *s* and *t*. You want to line up the nucleotides of the two sequences to find positions at which the two sequences are similar. However, these genes are so distant that the majority of the positions will be different. Nevertheless, conserved substrings between the two may be responsible for similar biological function. The score of an alignment is defined as the sum of the scores of each position of the alignment of the found substrings, the score of a match is +*m*, the score of a mismatch is -*μ*, and the score of a gap is -*σ*. A local alignment is simply an alignment between any substring of *s* and any substring of *t*, and you wish to find the highest-scoring local alignment. An example of the difference between alignment graphs for global and local alignments is provided below. Adding “free taxi rides” from the source to any point and from any point to the sink allows for alignments between any two substrings of *s* and *t* without having to deal with score penalties.



**Input Format.** The first line of the input contains *m* followed by *μ* followed by *σ* (separated by spaces), the second line of the input contains a DNA string *s*, and the third line of the input contains a DNA string *t*.

**Output Format.** The first line of the output should contain the score of an optimal local alignment between *s* and *t*, and the next two lines should contain a local alignment between substrings of s and *t* achieving this maximum score. Specifically, the second line should contain *s* with gaps placed appropriately, and the third line should contain *t* with gaps placed appropriately.

**Constraints.** |*s*| ≤ 1,000; |*t*| ≤ 1,000

**SAMPLE DATASET:**

Input:

1 1 2

GAGA

GAT

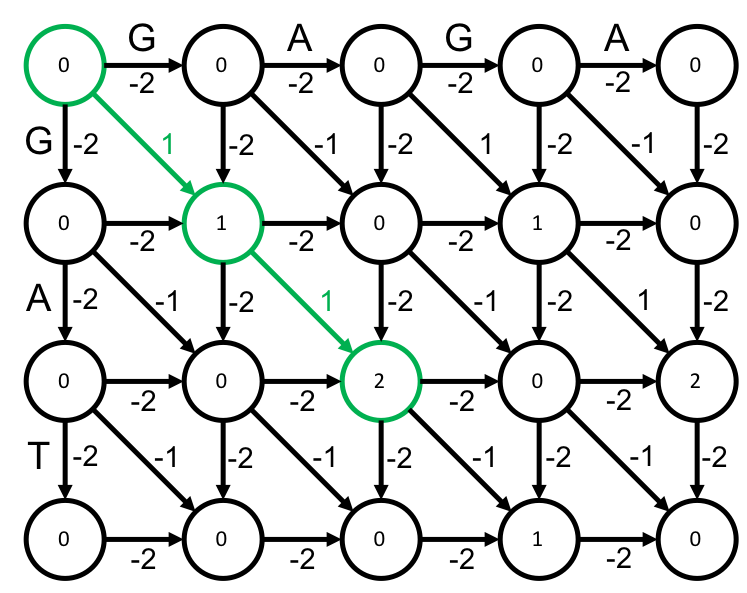
Output:

2

GA

GA

The highest score of a local alignment between **GA**GA and **GA**T is 2, and the above alignment (represented by the green path) achieves this maximum score.



**TEST DATASET 1:**

Input:

3 3 1

AGC

ATC

Output:

4

AG-C OR A-GC

A-TC AT-C

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. If the mismatch and indel penalties were switched somewhere in your code you would likely get a score of 5 with an alignment of:

AGC

ATC

**TEST DATASET 2:**

Input:

1 1 1

AT

AG

Output:

1

A

A

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.

**TEST DATASET 3:**

Input:

1 1 1

TAACG

ACGTG

Output:

3

ACG

ACG

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.

**TEST DATASET 4:**

Input:

3 2 1

CAGAGATGGCCG

ACG

Output:

6

CG

CG

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 or . If your code incorrectly sets the dynamic programming matrix dimensions to or it will not necessarily fail previous datasets since is the same as in all previous test datasets but it will fail this one.

**TEST DATASET 5:**

Input:

2 3 1

CTT

AGCATAAAGCATT

Output:

5

C-TT

CATT

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 4 except that in this dataset string *s* is shorter than string *t*.