1. Given A = GAATTC, B = GATTA, perform the global similarity alignment computation using the following score function, s(a, b) = 2 if a = b, s(a, b) = −1 if a = b, and s(a,−) = s(−, b) = −1. Print the computation matrix, indicate a path in the matrix that produce the optimal solution, and show the optimal global alignment.

Sol. Two sequences are given:

A = GAATTC
B = GATTA

Match = 2; Mismatch = -1; Gap = -1

	-	**G**	**A**	**A**	**T**	**T**	**C**
-	0	-1	-2	-3	-4	-5	-6
G	-1	2	1	0	-1	-2	-3
A	-2	1	4	3	2	1	0
T	-3	0	3	3	5	4	3
T	-4	-1	2	2	5	7	6
A	-5	-2	1	4	4	6	6

Paths for optimal alignments are indicated in blue

	-	**G**	**A**	**A**	**T**	**T**	**C**
-	0	-1	-2	-3	-4	-5	-6
G	-1	2	1	0	-1	-2	-3
A	-2	1	4	3	2	1	0
T	-3	0	3	3	5	4	3
T	-4	-1	2	2	5	7	6
A	-5	-2	1	4	4	6	6

Optimal global alignments:

GAATTC

G-ATTA

	-	**G**	**A**	**A**	**T**	**T**	**C**
-	0	-1	-2	-3	-4	-5	-6
G	-1	2	1	0	-1	-2	-3
A	-2	1	4	3	2	1	0
T	-3	0	3	3	5	4	3
T	-4	-1	2	2	5	7	6
A	-5	-2	1	4	4	6	6

GAATTC

GA-TTA

2. Given P = GCG, T = CTGAGAT where P is the pattern and T is the text, perform the fit similarity alignment computation (i.e., fitting P to T) using the following score function, s(a, b) = 2 if a = b, s(a, b) = −1 if a 6= b, and s(a,−) = s(−, b) = −1. Print the computation matrix, indicate a path in the matrix that produce the best fit, and show the alignment between P and a substring of T that produce the optimal solution.

Sol. Two sequences are given:

P = GCG
T = CTGAGAT

Match = 2; Mismatch = -1; Gap = -1

	-	C	T	G	A	G	A	T
-	0	← -1	← -2	-3	-4	-5	-6	-7
G	-1	-1	-2	↖ 0	-1	-2	-3	-4
C	-2	1	0	-1	↖ -1	-2	-3	-4
G	-3	0	0	2	1	↖ 1	← 0	← -1

Fit similarity alignment computation matrix (i.e., fitting P to T)

The alignment between P and a substring of T that produce the optimal solution:

CTGAGAT
- - GCG - -

3. Given A = GATACTTG, B = AATATGTA, perform the local similarity alignment computation using the following score function, s(a, b) = 2 if a = b, s(a, b) = −1 if a = b, and s(a,−) = s(−, b) = −1. Print the computation matrix, indicate a path in the matrix that produce the optimal solution, and show the aligment between a substring of A and a substring of B that produce the optimal solution.

Sol. Two sequences are given:

A = GATACTTG
B = AATATGTA

Match = 2; Mismatch = -1; Gap = -1

	-	**G**	**A**	**T**	**A**	**C**	**T**	**T**	**G**
-	0	0	0	0	0	0	0	0	0
A	0	0	2	1	2	1	0	0	0
A	0	0	2	1	3	2	1	0	0
T	0	0	1	4	3	2	4	3	2
A	0	0	2	3	6	5	4	3	2
T	0	0	1	4	5	5	7	6	5
G	0	2	1	3	4	4	6	6	8
T	0	1	1	3	3	3	6	8	7
A	0	0	3	2	5	4	5	7	7

Paths for optimal alignments are indicated in blue

ATACTTG
ATA - -TG

ATACTTG
ATA -T- G

ATACT-T
ATA-TGT

4. Generalize the algorithm for fitting one sequence into another to the case of g(k) = g_{ini} + kg_{ext}. Show why your algorithm is correct.

Sol. Let g(k) = g_{ini} + kg_{ext} for constants g_{ini} and g_{ext}.

$g_{ini} \geq 0$: gap initiation cost. $g_{ext} \geq 0$: gap extension cost.

To align sequences A and B. Let A = $a_1a_2 \ldots a_i$ B = $b_1b_2 \ldots b_j$

Any alignment of these two sequences is one of the following **three types**:

Type 1. $a_1\, a_2\, a_3\, a_4 \ldots a_i$
$b_1\, b_2\, b_3\, b_4 \ldots b_j$
alignment of A and B where characters a_i and b_j are aligned opposite each other. This includes both the case that $a_i = b_j$ and that $a_i \neq b_j$

Type 2. $a_1\, a_2\, a_3\, a_4 \ldots a_i$ ------
$b_1\, b_2\, b_3\, b_4 \ldots\ldots\ldots\ldots b_j$
alignment of A and B where character a_i is aligned to a character strictly to the left of character b_j. Therefore, the alignment ends with a gap in A.

Type 3. $a_1\, a_2\, a_3\, a_4 \ldots\ldots\ldots\ldots a_i$
$b_1\, b_2\, b_3\, b_4 \ldots b_j$ ------
alignment of A and B where character a_i is aligned to a character strictly to the right of character b_j. Therefore, the alignment ends with a gap in B.

Let us use the following notation:
• X(i, j) the max value of any alignment of **type 1**
• Y(i, j) the max value of any alignment of **type 2**
• Z(i, j) the max value of any alignment of **type 3**
• W(i, j) the max value of an alignment

Using the above notations, we can recursively define the alignment table. In the base conditions, we need to look at indel operations and assign the correct value: not only the weight of the spaces (g_{ext}), but also the weight of "opening the gap" (g_{ini}).
We will define 3 recurrence relations, one for each of X(i, j), Y(i, j) and Z(i, j). Each will be calculated from previously computed values. Take Y(i, j) for example. We are looking at alignments in which A ends to the left of B:

$a_1\, a_2\, a_3\, a_4 \ldots a_i$ ------
$b_1\, b_2\, b_3\, b_4 \ldots\ldots\ldots\ldots b_j$

The possible cases for the above alignment are the two given below:

1. It looked the same, i.e., A ended to the left of B.
Here, we only add "extension weight" to the value, forming new weight **Y(i, j – 1) + g_{ext}**
2. A and B ended at the same place (*type 1 alignment*).
In this case, we add both the "opening weight" gap and the "extension weight" gap, forming new weight **W(i, j – 1) + g_{ini} + g_{ext}.**

Taking the maximum of the two yields the value for E(i, j).

Calculating Z(i, j) and X(i, j) is done using similar arguments. W(i, j) is calculated by simply taking the maximum of the three. As in Global alignment, we search for the value V(n,m), and trace the alignment back using pointers created while filling the table.