Overview

For this homework you must compute two versions of sequence alignment.

- Maximizing the longest common subsequence (LCS)
- Optimizing the alignment when there is
  - +1 for an exact match
  - -1 for a mismatch
  - -1 for an inserted gap

We will work through an example for illustration, using strings

$X =$ `AATGGATTAG`
$Y =$ `CTCCCAGGTA`

and for the sake of this discussion, we will use a 1-index labeling of $X = X_1X_2 \ldots X_m$ and $Y = Y_1Y_2 \ldots Y_n$.

LCS

We will compute the length of the longest common subsequence using the following recursive formula. For strings X and Y with lengths $|X| = m$ and $|Y| = n$, we define $LCS[j][k]$ to be the longest common subsequence between the first j characters of X and the first k characters of Y. This can be computed with the following formula.

$$LCS[j][k] \;=\; \begin{cases} 0 & \text{if } j = 0 \text{ or } k = 0 \\ 1 + LCS[j-1][k-1] & \text{if } j, k > 0 \text{ and } X_j = Y_k \\ \max(LCS[j-1][k], LCS[j][k-1]) & \text{otherwise} \end{cases}$$

In affect, what we are interested in is the value $LCS[m][n]$ as if we have the first m characters of X then we have the entire string, and likewise for Y. However, the table of LCS values can be computed row-by-row starting with $j = 0$, then $j = 1$, and so forth. For our sample X and Y, the full table appears as follows (with the highlighted cells illustrating the subsequent reconstruction of the alignment that achieves the optimal).

			C	T	C	C	C	A	G	G	T	A	
		0	1	2	3	4	5	6	7	8	9	10	k
	0	0	0	0	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	0	0	1	1	1	1	1	
A	2	0	0	0	0	0	0	1	1	1	1	2	
T	3	0	0	1	1	1	1	1	1	1	2	2	
G	4	0	0	1	1	1	1	1	2	2	2	2	
G	5	0	0	1	1	1	1	1	2	3	3	3	
A	6	0	0	1	1	1	1	2	2	3	3	4	
T	7	0	0	1	1	1	1	2	2	3	4	4	
T	8	0	0	1	1	1	1	2	2	3	4	4	
A	9	0	0	1	1	1	1	2	2	3	4	5	
G	10	0	0	1	1	1	1	2	3	3	4	5	
	j												

From this table, we see that the longest common subsequence has length 5. We can reconstruct the actual sequence alignment that produces that common subsequence by examining the table starting with $j = m$ and $k = n$. That 5 must come from one of two cases. If $X_{10} = Y_{10}$ then it would be that the 5 was set as $1 + LCS[10][10]$. But in this case, since $X_{10} =$ `G` and $Y_{10} =$ `A`, the final recursive rule must have been used with $LCS[10][10] = \max(LCS[10][9], LCS[9][10])$. Examining the table, we see that this max was achieved by $LCS[9][10]$. So in our reconstruction of the sequence alignment, we will match the final `G` of X with an inserted gap at the end of Y, thus

```
X: ...g
Y: ...-
```

(Note: we use lowercase letters to denote unmatched characters.) Continuing, we see that the value of $LCS[9][10] = 5$ was set because $X_9 = Y_{10} =$ `A` and thus $LCS[9][10] = 1 + LCS[8][9]$ and that are alignment reconstruction will look something like

```
X: ...Ag
Y: ...A-
```

Likewise, $LCS[8][9] = 1 + LCS[7][8]$ as $X_8 = Y_9 =$ `T`, and our partial alignment becomes

```
X: ...TAg
Y: ...TA-
```

Continuing in this way, we determine that it must be that $LCS[7][8] = LCS[6][8] = LCS[5][8]$ and that $LCS[5][8] = 1 + LCS[4][7]$ given the match of $X_5 = Y_8 =$ `G`. Continuing in this way, we reach the complete alignment

```
-----aAtGGatTAg
ctccc-A-GG--TA-
```

Since not penalized for mismatches, we could further condense this alignment to

```
----aAtGGatTAg
ctcccA-GG--TA-
```

Scored Alignment

For our second scoring, we continue to have a +1 benefit for a match, but we introduce a -1 penalty for a gap or a -1 penalty for leaving two mismatched characters aligned. Our goal is to compute a score of an optimal alignment, and so we introduce the notation $OPT[j][k]$ as the optimal score that can be achieved when aligning the first j characters of X with the first k characters of Y. We claim the following recursive definitions.

$$OPT[j][k] = \begin{cases} -k & \text{if } j = 0 \\ -j & \text{if } k = 0 \\ 1 + OPT[j-1][k-1] & \text{if } j,k > 0 \text{ and } X_j = Y_k \\ \max(OPT[j-1][k], & \\ \quad OPT[j][k-1], & \\ \quad OPT[j-1][k-1]) - 1 & \text{otherwise} \end{cases}$$

Revisiting our example X and Y, this formula would produce the following table:

			C	T	C	C	C	A	G	G	T	A	
		0	1	2	3	4	5	6	7	8	9	10	k
	0	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	
A	1	-1	-1	-2	-3	-4	-5	-4	-5	-6	-7	-8	
A	2	-2	-2	-2	-3	-4	-5	-4	-5	-6	-7	-6	
T	3	-3	-3	-1	-2	-3	-4	-5	-5	-6	-5	-6	
G	4	-4	-4	-2	-2	-3	-4	-5	-4	-4	-5	-6	
G	5	-5	-5	-3	-3	-3	-4	-5	-4	-3	-4	-5	
A	6	-6	-6	-4	-4	-4	-4	-3	-4	-4	-4	-3	
T	7	-7	-7	-5	-5	-5	-5	-4	-4	-5	-3	-4	
T	8	-8	-8	-6	-6	-6	-6	-5	-5	-5	-4	-4	
A	9	-9	-9	-7	-7	-7	-7	-5	-6	-6	-5	-3	
G	10	-10	-10	-8	-8	-8	-8	-6	-4	-5	-6	-4	
	j												

Again, we can later reconstruct the actual alignment by considering, starting with $OPT[m][n]$, how each cell depends upon the cell for one of the subproblems, which designates when we have aligned matches, aligned mismatches, or inserted gaps. The highlighted path in this table shows one possible way to achieve the optimal score, corresponding to the following actual alignment:

```
aaT-ggA-tTAg
-cTcccAggTA-
```

Your Task

We will be providing you with a *different* choice of strings X and Y and you must produce both the full table and the subsequently reconstructed alignment for each of the two scoring metrics: LCS and OPT.