AI Lab 03
Section C

Task 01: The edit distance between two strings refers to the minimum number of character insertions, deletions, and substitutions required to change one string to the other. For example, the edit distance between "kitten" and "sitting" is three: substitute the "k" for "s", substitute the "e" for "i", and append a "g".

Write a Python program to compute the optimal sequence alignment of two DNA strings using edit distance.

Example:

We align the two sequences, but we are permitted to *insert gaps* in either sequence (e.g., to make them have the same length). We pay a penalty for each gap that we insert and also for each pair of characters that mismatch in the final alignment. Intuitively, these penalties model the relative likeliness of point mutations arising from deletion/insertion and substitution. We produce a numerical score according to the following table, which is widely used in biological applications:

operation	***cost***
insert a gap	2
align two characters that mismatch	1
align two characters that match	0

Here are two possible alignments of the strings *x* = "AACAGTTACC" and *y* = "TAAGGTCA":

```
 x   y  cost          x   y  cost
------------         ------------
 A   T   1            A   T   1
 A   A   0            A   A   0
 C   A   1            C   -   2
 A   G   1            A   A   0
 G   G   0            G   G   0
 T   T   0            T   G   1
 T   C   1            T   T   0
 A   A   0            A   -   2
 C   -   2            C   C   0
 C   -   2            C   A   1
        ---                  ---
         8                    7
```

The first alignment has a score of 8, while the second one has a score of 7. The *edit-distance* is the score of the best possible alignment between the two genetic sequences over all possible alignments. In this example, the second alignment is in fact optimal, so the edit-distance between the two strings is 7.

We can compute opt[i][j].cost by taking the minimum of three quantities:

opt[i][j].cost = min { opt[i+1][j+1].cost + 0/1, opt[i+1][j].cost + 2, opt[i][j+1].cost + 2 }

Task 02: Given an unbalanced bracket sequence of **'('** and **')'**, convert it into a balanced sequence by adding the minimum number of **'('** at the beginning of the string and **')'** at the end of the string by dynamic programming.

Example:

Input: (a+b(c)

Output: (a+b(c))

Task 03: Suppose we have an optimal alignment for two sequences S1…n and T1…m in which Si matches Tj. The key insight is that this optimal alignment is composed of an optimal alignment between (S1,…,Si−1) and (T1,…,Ti−1) and an optimal alignment between (Si+1,…,Sn) and (Tj+1,…,Tm). This follows from a cut-and-paste argument: if one of these partial alignments is suboptimal, then we cut-and-paste a better alignment in place of the suboptimal one. This achieves a higher score of the overall alignment and thus contradicts the optimality of the initial global alignment. In other words, every sub-path in an optimal path must also be optimal. Notice that the scores are additive, so the score of the overall alignment equals the addition of the scores of the alignments of the subsequences. This implicitly assumes that the sub-problems of computing the optimal scoring alignments of the subsequences are independent. We need to biologically motivate that such an assumption leads to meaningful results.

To solve the problem, we will traverse the matrix column by column computing the optimal score for each alignment sub-problem by considering the four possibilities:

- Sequence S has a gap at the current alignment position.
- Sequence T has a gap at the current alignment position.
- There is a mutation (nucleotide substitution) at the current position.
- There is a match at the current position.

We then use the possibility that produces the maximum score. We express this mathematically by the recursive formula for Fi,j:

$$F(0,0) = 0$$

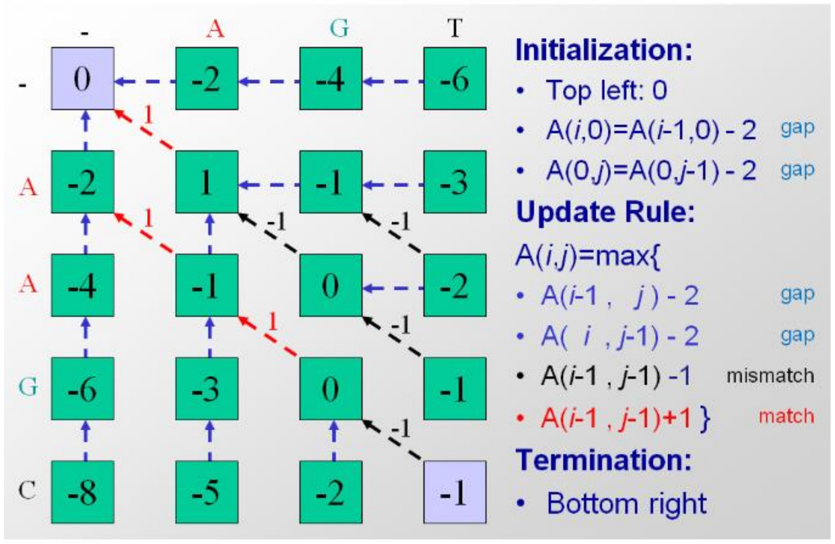
$$Initialization : F(i,0) = F(i-1,0) - d$$

$$F(0,j) = F(0,j-1) - d$$

$$\text{Iteration :} \quad F(i,j) = \max \begin{cases} F(i-1,j) - d \text{ insert gap in S} \\ F(i,j-1) - d \text{ insert gap in T} \\ F(i-1,j-1) + s(x_i, y_j) \text{ match or mutation} \end{cases}$$

Termination : Bottom right

Example:



Back tacking:

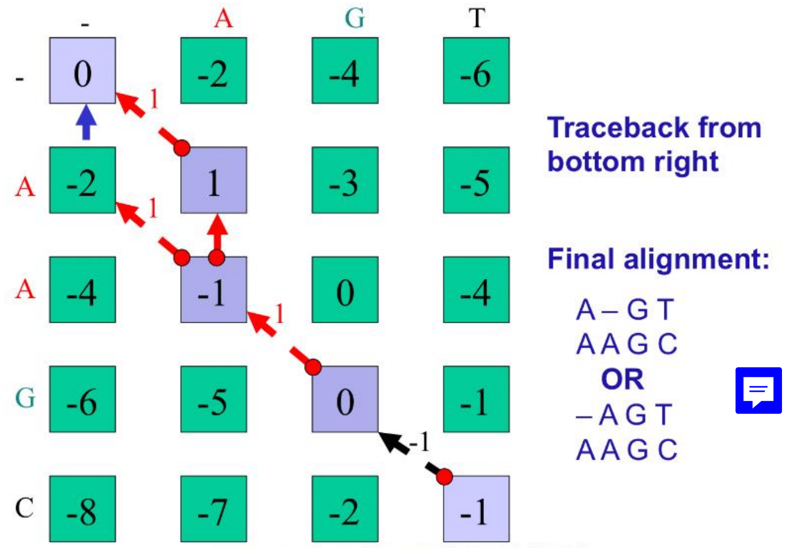


Figure 2.11: (Example) Tracing the optimal alignment

Write dynamic programming solution for this problem.