AI Lab 03
Section D

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 edit distance between two given strings.

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 subproblem 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:

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

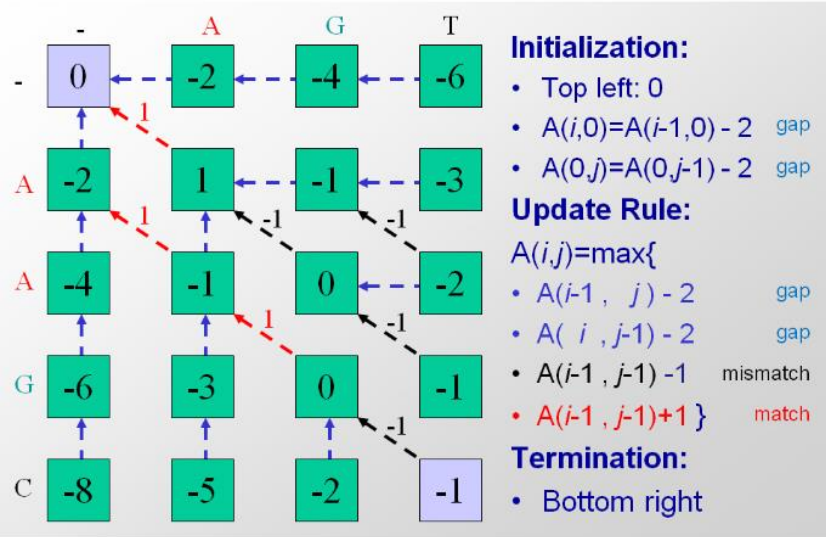
$$Initialization : F(i,0) = F(i-1,0) - d$$

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

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

$$\text{Termination : Bottom right}$$

Example:



Back tacking:

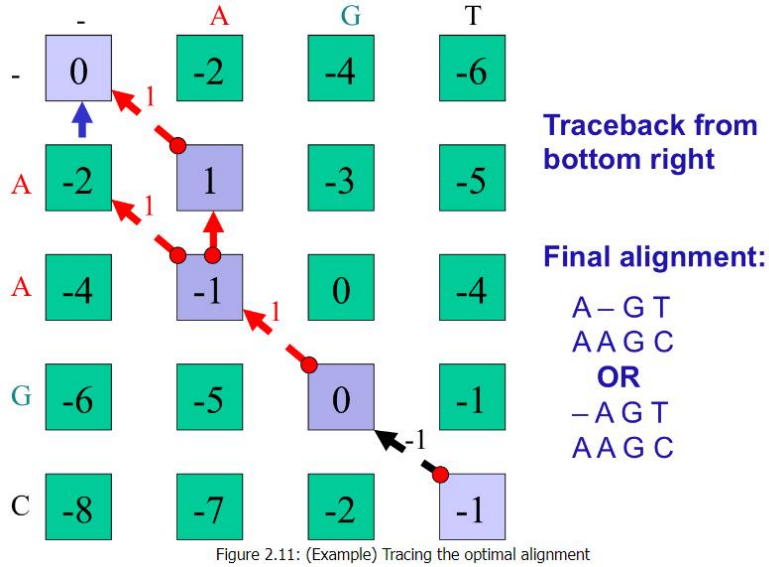


Figure 2.11: (Example) Tracing the optimal alignment

Task 04:

Problem Statement: Given an array of available denominations of coin and one target price. Find the minimum number of coins required to pay the same.

Solution: Here, you can start the solution with sum = N cents. In each iteration, find the minimum coins required by dividing the original problem into subproblems.

Consider a coin from { 1, c2,…cm} and reduce the sum repeatedly depending upon the coin of the denomination you choose. You have to repeat the same process until N becomes 0, and at this point, you found your solution.

Example 1

Input: coin[] = [25, 10, 5], K = 30, Output: 2

Explanation: Minimum 2 coins required: we can use one coin of 25 and one coin of 5.

Example 2

Input: coin[] = [9, 6, 5, 1], K = 13, Output: 3

Explanation: Minimum 3 coins required: we can use two coins of 6 and one coin of 1.

Example 3

Input: coin[] = [1, 3, 5, 7], K = 18, Output: 4

Explanation: Minimum 4 coins required. We can use any one of these combinations to provide change using 4 coins: (7, 7, 3, 1),(5, 5, 5, 3), and (7, 5, 5, 1).

Problem Logic:

Base Case:

If N==**0**, then **0** coins are considered

Case **1**:

If N>**0** then,
minCoinsRequired(N, coins[**0**,**1**,**2**..m-**1**]) = min(**1**+ minCoinsRequired(N-coins[i], coins[**0**…m-**1**]))
where i is from 0 to m-1 and coin[i] <= N