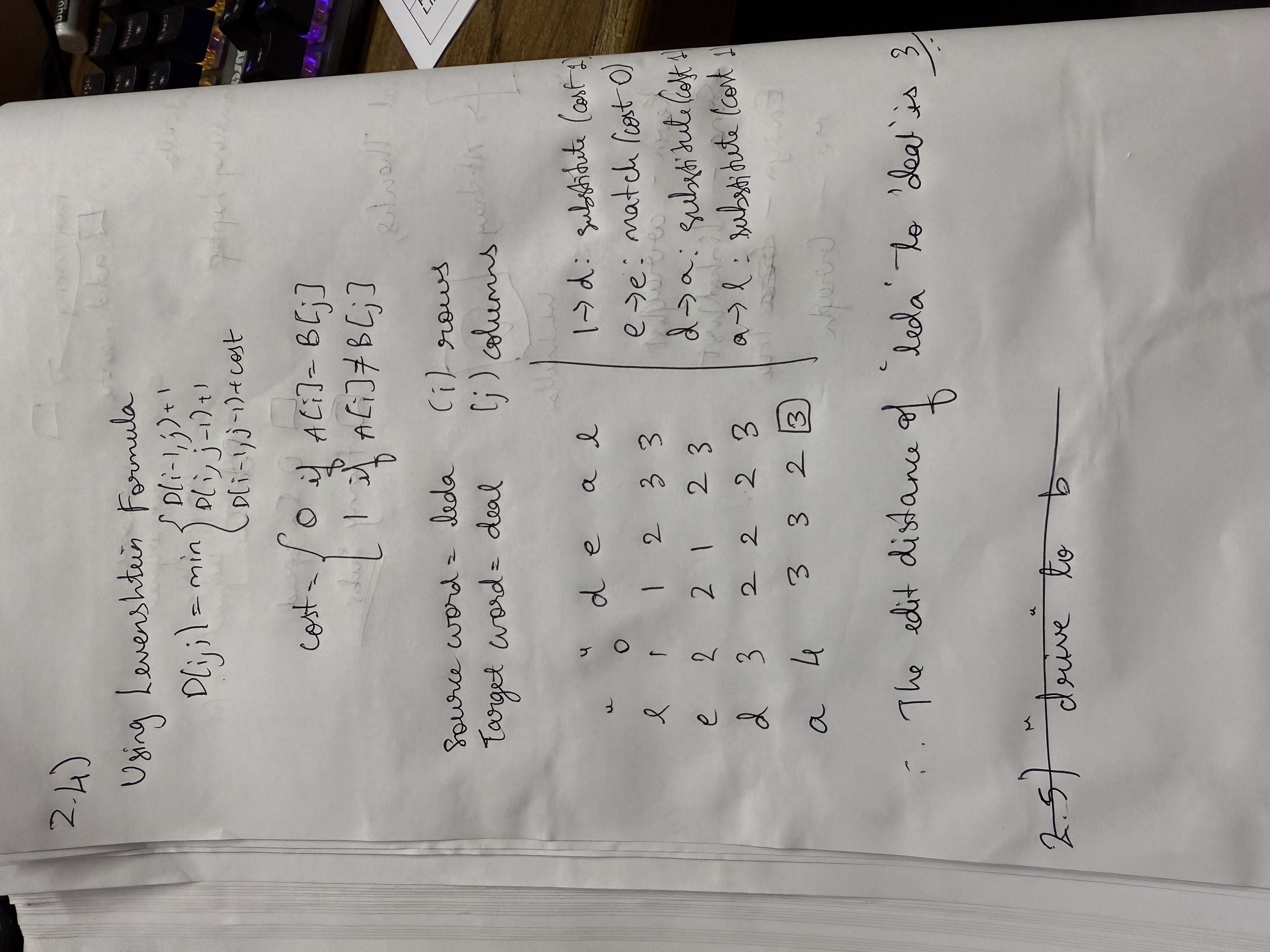
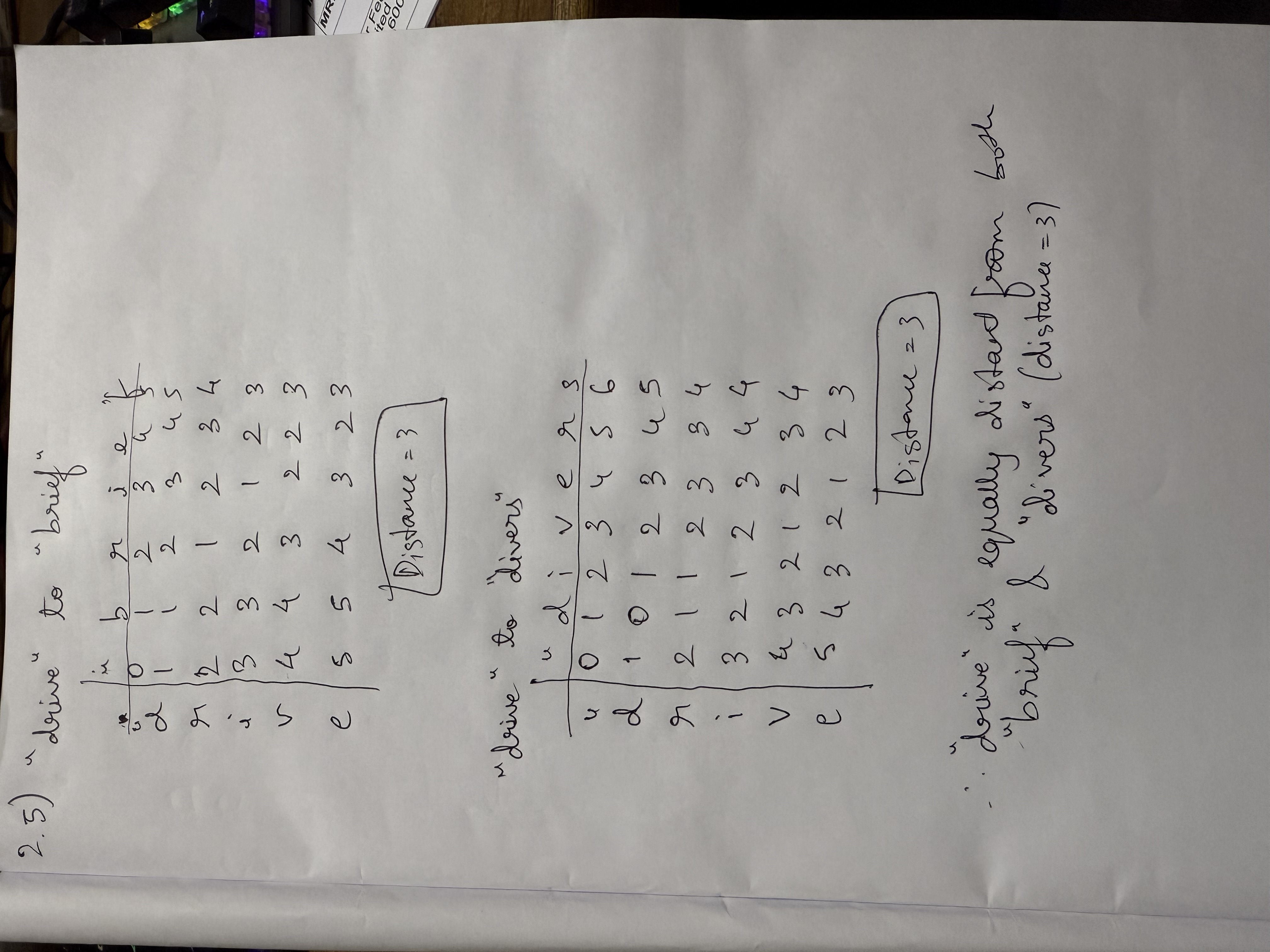
# Lab2: Edit Distance and Applications

Date: 30 June 2025  
R Abhijit Srivathsan  
2448044





Program Question:  
1. Implement a minimum edit distance algorithm and use your hand-computed results to check your code.  
2. Implement Sequence Alignment using the Needleman-Wunsch algorithm for two input sequences.

## Draft Plan:

The goal is to create two Python programs that take user input and solve string comparison problems:  
- One using the Levenshtein Distance to calculate how different two strings are.  
- The second using a global sequence alignment algorithm to match two sequences by introducing gaps.  
  
Both programs will be menu-driven and allow the user to enter strings or sequences and view the result.

## Program Description:

These programs are designed to help understand two fundamental NLP and bioinformatics techniques:  
- Edit Distance Program: Calculates the Levenshtein Distance between two strings using NLTK.  
- Sequence Alignment Program: Aligns two DNA or text sequences using the Needleman–Wunsch algorithm.  
  
Both are interactive and run in the terminal.

## Program Logic:

- Edit Distance:  
 - Uses the edit\_distance function from the NLTK library.  
 - Input: Two strings from the user.  
 - Output: Minimum number of edits required to convert one string into another.  
 - Uses basic string operations and a standard NLP utility.  
  
- Sequence Alignment:  
 - Implements the Needleman–Wunsch algorithm.  
 - Uses nested loops to build a scoring matrix and backtracking for alignment.  
 - Configurable match, mismatch, and gap penalty values.  
 - Aligns sequences with '-' to denote insertions/deletions.  
  
Both programs use simple data types like strings and lists, and follow structured logic.

## Program:

import nltk

from nltk.metrics import edit\_distance

def compute\_edit\_distance(s1, s2):

    """Compute the Levenshtein edit distance between two strings using NLTK."""

    return edit\_distance(s1, s2)

def needleman\_wunsch(seq1, seq2, match\_score=1, mismatch\_score=-1, gap\_penalty=-2):

    """Perform global sequence alignment (Needleman–Wunsch) on two sequences."""

    len1, len2 = len(seq1), len(seq2)

    # Initialize scoring matrix

    score = [[0] \* (len2 + 1) for \_ in range(len1 + 1)]

    for i in range(1, len1 + 1):

        score[i][0] = score[i-1][0] + gap\_penalty

    for j in range(1, len2 + 1):

        score[0][j] = score[0][j-1] + gap\_penalty

    # Fill the scoring matrix

    for i in range(1, len1 + 1):

        for j in range(1, len2 + 1):

            match = score[i-1][j-1] + (match\_score if seq1[i-1] == seq2[j-1] else mismatch\_score)

            delete = score[i-1][j] + gap\_penalty

            insert = score[i][j-1] + gap\_penalty

            score[i][j] = max(match, delete, insert)

    # Traceback to build alignment

    align1, align2 = "", ""

    i, j = len1, len2

    while i > 0 and j > 0:

        current = score[i][j]

        if current == score[i-1][j-1] + (match\_score if seq1[i-1] == seq2[j-1] else mismatch\_score):

            align1 = seq1[i-1] + align1

            align2 = seq2[j-1] + align2

            i -= 1

            j -= 1

        elif current == score[i-1][j] + gap\_penalty:

            align1 = seq1[i-1] + align1

            align2 = "-" + align2

            i -= 1

        else:

            align1 = "-" + align1

            align2 = seq2[j-1] + align2

            j -= 1

    # Finish tracing up to the matrix origin

    while i > 0:

        align1 = seq1[i-1] + align1

        align2 = "-" + align2

        i -= 1

    while j > 0:

        align1 = "-" + align1

        align2 = seq2[j-1] + align2

        j -= 1

    return align1, align2

def main():

    while True:

        print("\nMenu:")

        print("1. Compute Edit Distance")

        print("2. Perform Sequence Alignment")

        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ").strip()

        if choice == "1":

            s1 = input("Enter the first string: ").strip()

            s2 = input("Enter the second string: ").strip()

            dist = compute\_edit\_distance(s1, s2)

            print(f"Edit distance between '{s1}' and '{s2}' is {dist}.")

        elif choice == "2":

            seq1 = input("Enter sequence A: ").strip().upper()

            seq2 = input("Enter sequence B: ").strip().upper()

            align1, align2 = needleman\_wunsch(seq1, seq2)

            print("\nAligned Sequences:")

            print(align1)

            print(align2)

        elif choice == "3":

            print("Exiting program. Goodbye!")

            break

        else:

            print("Invalid choice. Please enter 1, 2, or 3.")

if \_\_name\_\_ == "\_\_main\_\_":

    # Ensure NLTK resources are available

    try:

        nltk.data.find('tokenizers/punkt')

    except LookupError:

        nltk.download('punkt')

    main()

## Test Cases with Actual and Expected I/O:

### Edit Distance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case | String 1 | String 2 | Expected Distance | Actual Output |
| 1 | kitten | sitting | 3 | 3 |
| 2 | leda | deal | 3 | 3 |
| 3 | flaw | lawn | 2 | 2 |

### Sequence Alignment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case | Seq A | Seq B | Aligned A | Aligned B |
| 1 | GATTACA | GCATGCU | GATTACA | GCATGCU |
| 2 | TAGCT... | TAGCT... | TAGCT... | TAGCT... |
| 3 | AGGCT... | TAGCT... | AGGCTATCAC...--TG-CC--C | TAGCTATCA...GGTCGATTTGCCCGAC |

## Evaluation Comments:

(To be filled by faculty)