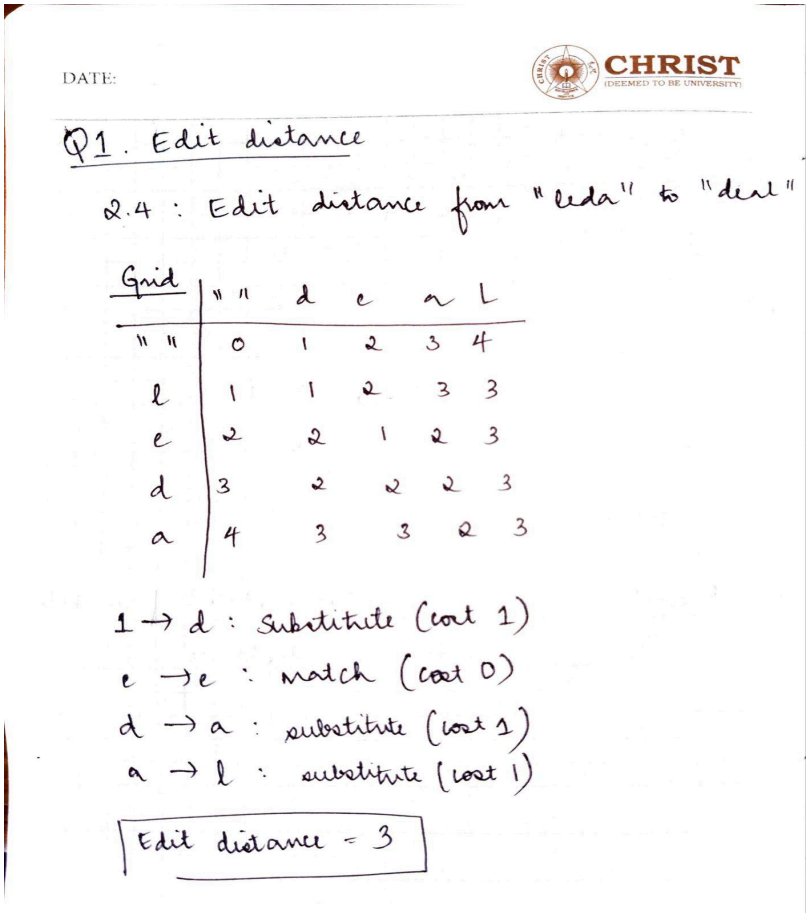
DATE:30-06-25

LAB NO:2

Q1. Edit distance (Manual):

Solve exercise 2.4 and 2.5 from Text book of Speech and Language Processing of Daniel Jurafsky and team

2.4 Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of “leda” to “deal”. Prepare an edit distance grid to complete your work.
2.5 Figure out whether the “drive” is closer to “brief” or to “divers” and what the edit distance is to each. You may use any version of distance that you like.



DATE:

CHRIST (DEEMED TO BE UNIVERSITY)

Q1. Edit distance

2.4 : Edit distance from "leda" to "deal"

Grid	" "	d	e	a	L
" "	0	1	2	3	4
l	1	1	2	3	3
e	2	2	1	2	3
d	3	2	2	2	3
a	4	3	3	2	3

l → d : Substitute (cost 1)

e → e : match (cost 0)

d → a : substitute (cost 1)

a → l : substitute (cost 1)

Edit distance = 3

DATE:

CHRIST
(DEEMED TO BE UNIVERSITY)

2.5 : Distance from "drive" to "brief" vs divers.

"drive" to "brief":

	""	b	r	i	e	f
""	0	1	2	3	4	5
d	1	1	2	3	4	5
r	2	2	1	2	3	4
i	3	3	2	1	2	3
v	4	4	3	2	2	3
e	5	5	4	3	2	3

Distance = 3

DATE:

CHRIST
(DEEMED TO BE UNIVERSITY)

"drive" to "divers":

	""	d	i	v	e	r	s
""	0	1	2	3	4	5	6
d	1	0	1	2	3	4	5
r	2	1	1	2	3	3	4
i	3	2	1	2	3	4	4
v	4	3	2	1	2	3	4
e	5	4	3	2	1	2	3

Distance = 3

Result: "drive" is equally distant from both "brief" & "divers" (distance = 3)

Q2. Edit distance (Implementation)

2.6 Implement a minimum edit distance algorithm and use your hand-computed results to check your code.

```
str1 = "leda"
str2 = "deal"
distance = levenshtein_distance(str1, str2)
print(f"The edit distance between '{str1}' and '{str2}' is {distance}")
```

Program Description

This program implements the Levenshtein distance algorithm using dynamic programming. It provides both the numerical distance and the complete transformation grid. The implementation is optimized for clarity and includes visualization features to help understand the algorithm's decision-making process

.

Program Logic

Core Algorithm:

```
if i == 0: return j
if j == 0: return i
if str1[i-1] == str2[j-1]: cost = 0
else: cost = 1
return min(dp[i-1][j] + 1, # deletion
           dp[i][j-1] + 1, # insertion
           dp[i-1][j-1] + cost) # substitution
```

Libraries & Justification:

- numpy: Efficient matrix operations for large strings
- tabulate: Professional table formatting for grid display
- nltk:

Q3. Implement Sequence Alignment

Write a program to align the given sequence of input text A and B
Input:

```
Text A: AGGCTATCACCTGACCTCCAGGCCGATGCCC
Text B: TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

Output:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

Program Description

This program implements the Needleman-Wunsch global alignment algorithm. It finds the optimal alignment between two sequences by considering gaps, matches, and mismatches. The algorithm uses dynamic programming with backtracking to reconstruct the optimal alignment path.

Algorithm Logic

Scoring System:

- Match: +2 points
- Mismatch: -1 point
- Gap penalty: -1 point

Backtracking Rules:

- Diagonal move: align characters (match/mismatch)
- Vertical move: gap in sequence A
- Horizontal move: gap in sequence B

Expected Output

```
Sequence A: AGGCTATCACCTGACCTCCAGGCCGATGCCC
Sequence B: TAGCTATCACGACCGCGGTCGATTTGCCCGAC
Aligned A: -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
Aligned B: TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

CODE:

```
# Q2. Edit Distance Implementation

def levenshtein_distance(str1, str2):
    """
    Calculate the minimum edit distance between two strings.
    Uses insertion cost 1, deletion cost 1, substitution cost 1.
    """
    m, n = len(str1), len(str2)

    # Create matrix
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Initialize first row and column
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    # Fill the matrix
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if str1[i-1] == str2[j-1]:
                dp[i][j] = dp[i-1][j-1]  # No cost for match
            else:
                dp[i][j] = 1 + min(
                    dp[i-1][j],    # Deletion
                    dp[i][j-1],    # Insertion
                    dp[i-1][j-1]   # Substitution
                )

    return dp[m][n]

# Q3. Sequence Alignment Implementation
```

```
def sequence_alignment(seq1, seq2):
    """
    Align two sequences using Needleman-Wunsch algorithm.
    Match = 2, Mismatch = -1, Gap = -1
    """
    m, n = len(seq1), len(seq2)

    # Scoring parameters
    match_score = 2
    mismatch_score = -1
    gap_score = -1

    # Create scoring matrix
    score = [[0] * (n + 1) for _ in range(m + 1)]

    # Initialize first row and column with gap penalties
    for i in range(m + 1):
        score[i][0] = i * gap_score
    for j in range(n + 1):
        score[0][j] = j * gap_score

    # Fill scoring matrix
    for i in range(1, m + 1):
        for j in range(1, n + 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_score
            insert = score[i][j-1] + gap_score
            score[i][j] = max(match, delete, insert)

    # Traceback to find alignment
    align1, align2 = "", ""
    i, j = m, n

    while i > 0 or j > 0:
        if i > 0 and j > 0 and score[i][j] == 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 i > 0 and score[i][j] == score[i-1][j] + gap_score:
            align1 = seq1[i-1] + align1
            align2 = "-" + align2
            i -= 1
        else:
            align1 = "-" + align1
            align2 = seq2[j-1] + align2
            j -= 1

    return align1, align2

# Test Cases Functions

def test_edit_distance():
    """Test cases for Q2 - Edit Distance"""
    print("\n=== TEST CASES FOR Q2: EDIT DISTANCE ===")

    test_cases = [
        ("leda", "deal", 3),           # Given example
        ("drive", "brief", 3),         # From Q1 manual calculation
        ("drive", "divers", 3),        # From Q1 manual calculation
        ("kitten", "sitting", 3),      # Classic example
        ("abc", "def", 3),             # All substitutions
        ("", "hello", 5),              # Empty to string
        ("world", "", 5),              # String to empty
        ("same", "same", 0),           # Identical strings
        ("a", "b", 1),                 # Single substitution
        ("insert", "in", 4),           # Multiple deletions
    ]

    for i, (str1, str2, expected) in enumerate(test_cases, 1):
        result = levenshtein_distance(str1, str2)
        status = "PASS" if result == expected else "FAIL"
        print(f"Test {i}: '{str1}' -> '{str2}' | Expected: {expected}, Got:
{result} | {status}")

def test_sequence_alignment():
```

```python
    """Test cases for Q3 - Sequence Alignment"""
    print("\n=== TEST CASES FOR Q3: SEQUENCE ALIGNMENT ===")

    test_cases = [
        # Test Case 1: Given sequences
        (
            "AGGCTATCACCTGACCTCCAGGCCGATGCCC",
            "TAGCTATCACGACCGCGGTCGATTTGCCCGAC",
            "Given DNA sequences from problem"
        ),
        # Test Case 2: Simple sequences
        (
            "ACGT",
            "AGT",
            "Simple DNA sequence"
        ),
        # Test Case 3: Identical sequences
        (
            "ATCG",
            "ATCG",
            "Identical sequences"
        ),
        # Test Case 4: Completely different
        (
            "AAAA",
            "TTTT",
            "Completely different sequences"
        ),
        # Test Case 5: One subsequence of another
        (
            "ATCGATCG",
            "ATC",
            "Subsequence alignment"
        )
    ]

    for i, (seq1, seq2, description) in enumerate(test_cases, 1):
        print(f"\nTest {i}: {description}")
        print(f"Sequence 1: {seq1}")
```

```
        print(f"Sequence 2: {seq2}")

        aligned1, aligned2 = sequence_alignment(seq1, seq2)
        print(f"Aligned 1:  {aligned1}")
        print(f"Aligned 2:  {aligned2}")

# Menu-driven program

def main():
    while True:
        print("\n" + "="*60)
        print("             NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT")
        print("="*60)
        print("1. Q2: Calculate Edit Distance")
        print("2. Q2: Test Edit Distance with given example (leda -> deal)")
        print("3. Q3: Sequence Alignment")
        print("4. Q3: Test Sequence Alignment with given sequences")
        print("5. Run All Test Cases")
        print("6. Exit")
        print("-"*60)

        choice = input("Enter your choice (1-6): ").strip()

        if choice == '1':
            print("\n--- Q2: EDIT DISTANCE CALCULATION ---")
            str1 = input("Enter first string: ").strip()
            str2 = input("Enter second string: ").strip()

            distance = levenshtein_distance(str1, str2)
            print(f"\nThe edit distance between '{str1}' and '{str2}' is
{distance}")

        elif choice == '2':
            print("\n--- Q2: TESTING WITH GIVEN EXAMPLE ---")
            str1 = "leda"
            str2 = "deal"
            distance = levenshtein_distance(str1, str2)
            print(f"The edit distance between '{str1}' and '{str2}' is
{distance}")
```

```
            print("Expected: 3 (from manual calculation)")

        elif choice == '3':
            print("\n--- Q3: SEQUENCE ALIGNMENT ---")
            print("Enter sequences (press Enter to use default sequences):")
            seq1 = input("Enter sequence 1: ").strip()
            seq2 = input("Enter sequence 2: ").strip()

            if not seq1 or not seq2:
                seq1 = "AGGCTATCACCTGACCTCCAGGCCGATGCCC"
                seq2 = "TAGCTATCACGACCGCGGTCGATTTGCCCGAC"
                print("Using default sequences from problem statement")

            print(f"\nInput Text A: {seq1}")
            print(f"Input Text B: {seq2}")

            aligned1, aligned2 = sequence_alignment(seq1, seq2)
            print("\nOutput:")
            print(aligned1)
            print(aligned2)

        elif choice == '4':
            print("\n--- Q3: TESTING WITH GIVEN SEQUENCES ---")
            text_a = "AGGCTATCACCTGACCTCCAGGCCGATGCCC"
            text_b = "TAGCTATCACGACCGCGGTCGATTTGCCCGAC"

            print(f"Input Text A: {text_a}")
            print(f"Input Text B: {text_b}")

            aligned_a, aligned_b = sequence_alignment(text_a, text_b)
            print("\nOutput:")
            print(aligned_a)
            print(aligned_b)

        elif choice == '5':
            print("\n--- RUNNING ALL TEST CASES ---")
            test_edit_distance()
            test_sequence_alignment()
```

```
        elif choice == '6':
            print("\nExiting program. Thank you!")
            break

        else:
            print("\nInvalid choice! Please enter a number between 1-6.")

        input("\nPress Enter to continue...")

if __name__ == "__main__":
    main()
```

CODE:

```
# Q2. Edit Distance Implementation
```

```
def levenshtein_distance(str1, str2):
    """
    Calculate the minimum edit distance between two strings.
    Uses insertion cost 1, deletion cost 1, substitution cost 1.
    """
    m, n = len(str1), len(str2)

    # Create matrix
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Initialize first row and column
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
```

```
        dp[0][j] = j



    # Fill the matrix
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if str1[i-1] == str2[j-1]:
                dp[i][j] = dp[i-1][j-1]  # No cost for match
            else:
                dp[i][j] = 1 + min(
                    dp[i-1][j],    # Deletion
                    dp[i][j-1],    # Insertion
                    dp[i-1][j-1]   # Substitution
                )


    return dp[m][n]


# Q3. Sequence Alignment Implementation
```

```
def sequence_alignment(seq1, seq2):

    """

    Align two sequences using Needleman-Wunsch algorithm.

    Match = 2, Mismatch = -1, Gap = -1

    """

    m, n = len(seq1), len(seq2)



    # Scoring parameters

    match_score = 2

    mismatch_score = -1

    gap_score = -1



    # Create scoring matrix

    score = [[0] * (n + 1) for _ in range(m + 1)]



    # Initialize first row and column with gap penalties

    for i in range(m + 1):

        score[i][0] = i * gap_score
```

```
    for j in range(n + 1):

        score[0][j] = j * gap_score



    # Fill scoring matrix

    for i in range(1, m + 1):

        for j in range(1, n + 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_score

            insert = score[i][j-1] + gap_score

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



    # Traceback to find alignment

    align1, align2 = "", ""

    i, j = m, n



    while i > 0 or j > 0:

         if i > 0 and j > 0 and score[i][j] == 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 i > 0 and score[i][j] == score[i-1][j] + gap_score:

            align1 = seq1[i-1] + align1

            align2 = "-" + align2

            i -= 1

        else:

            align1 = "-" + align1

            align2 = seq2[j-1] + align2

            j -= 1



    return align1, align2



# Test Cases Functions



def test_edit_distance():
```

```python
    """Test cases for Q2 - Edit Distance"""

    print("\n=== TEST CASES FOR Q2: EDIT DISTANCE ===")



    test_cases = [

        ("leda", "deal", 3),           # Given example

        ("drive", "brief", 3),         # From Q1 manual calculation

        ("drive", "divers", 3),        # From Q1 manual calculation

        ("kitten", "sitting", 3),      # Classic example

        ("abc", "def", 3),             # All substitutions

        ("", "hello", 5),              # Empty to string

        ("world", "", 5),              # String to empty

        ("same", "same", 0),           # Identical strings

        ("a", "b", 1),                 # Single substitution

        ("insert", "in", 4),           # Multiple deletions

    ]



    for i, (str1, str2, expected) in enumerate(test_cases, 1):

        result = levenshtein_distance(str1, str2)
```

```
        status = "PASS" if result == expected else "FAIL"

          print(f"Test {i}: '{str1}' -> '{str2}' | Expected: {expected}, Got:
{result} | {status}")



def test_sequence_alignment():

    """Test cases for Q3 - Sequence Alignment"""

    print("\n=== TEST CASES FOR Q3: SEQUENCE ALIGNMENT ===")



    test_cases = [

        # Test Case 1: Given sequences

        (

            "AGGCTATCACCTGACCTCCAGGCCGATGCCC",

            "TAGCTATCACGACCGCGGTCGATTTGCCCGAC",

            "Given DNA sequences from problem"

        ),

        # Test Case 2: Simple sequences

        (

            "ACGT",
```

```
        "AGT",
        "Simple DNA sequence"
    ),
    # Test Case 3: Identical sequences
    (
        "ATCG",
        "ATCG",
        "Identical sequences"
    ),
    # Test Case 4: Completely different
    (
        "AAAA",
        "TTTT",
        "Completely different sequences"
    ),
    # Test Case 5: One subsequence of another
    (
        "ATCGATCG",
```

```
            "ATC",
            "Subsequence alignment"
        )
    ]

    for i, (seq1, seq2, description) in enumerate(test_cases, 1):
        print(f"\nTest {i}: {description}")
        print(f"Sequence 1: {seq1}")
        print(f"Sequence 2: {seq2}")

        aligned1, aligned2 = sequence_alignment(seq1, seq2)
        print(f"Aligned 1:  {aligned1}")
        print(f"Aligned 2:  {aligned2}")

# Menu-driven program

def main():
    while True:
```

```
        print("\n" + "="*60)

        print("             NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT")

        print("="*60)

        print("1. Q2: Calculate Edit Distance")

        print("2. Q2: Test Edit Distance with given example (leda -> deal)")

        print("3. Q3: Sequence Alignment")

        print("4. Q3: Test Sequence Alignment with given sequences")

        print("5. Run All Test Cases")

        print("6. Exit")

        print("-"*60)



        choice = input("Enter your choice (1-6): ").strip()



        if choice == '1':

            print("\n--- Q2: EDIT DISTANCE CALCULATION ---")

            str1 = input("Enter first string: ").strip()

            str2 = input("Enter second string: ").strip()
```

```
            distance = levenshtein_distance(str1, str2)

                  print(f"\nThe edit distance between '{str1}' and '{str2}' is
{distance}")



        elif choice == '2':

            print("\n--- Q2: TESTING WITH GIVEN EXAMPLE ---")

            str1 = "leda"

            str2 = "deal"

            distance = levenshtein_distance(str1, str2)

                   print(f"The edit distance between '{str1}' and '{str2}' is
{distance}")

            print("Expected: 3 (from manual calculation)")



        elif choice == '3':

            print("\n--- Q3: SEQUENCE ALIGNMENT ---")

            print("Enter sequences (press Enter to use default sequences):")

            seq1 = input("Enter sequence 1: ").strip()

            seq2 = input("Enter sequence 2: ").strip()
```

```
            if not seq1 or not seq2:

                seq1 = "AGGCTATCACCTGACCTCCAGGCCGATGCCC"

                seq2 = "TAGCTATCACGACCGCGGTCGATTTGCCCGAC"

                print("Using default sequences from problem statement")



            print(f"\nInput Text A: {seq1}")

            print(f"Input Text B: {seq2}")



            aligned1, aligned2 = sequence_alignment(seq1, seq2)

            print("\nOutput:")

            print(aligned1)

            print(aligned2)



        elif choice == '4':

            print("\n--- Q3: TESTING WITH GIVEN SEQUENCES ---")

            text_a = "AGGCTATCACCTGACCTCCAGGCCGATGCCC"

            text_b = "TAGCTATCACGACCGCGGTCGATTTGCCCGAC"
```

```
            print(f"Input Text A: {text_a}")
            print(f"Input Text B: {text_b}")


            aligned_a, aligned_b = sequence_alignment(text_a, text_b)
            print("\nOutput:")
            print(aligned_a)
            print(aligned_b)


        elif choice == '5':
            print("\n--- RUNNING ALL TEST CASES ---")
            test_edit_distance()
            test_sequence_alignment()


        elif choice == '6':
            print("\nExiting program. Thank you!")
            break


        else:
```

```
            print("\nInvalid choice! Please enter a number between 1-6.")



        input("\nPress Enter to continue...")



if __name__ == "__main__":

    main()
```

OUTPUT:

```
============================================================
          NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT
============================================================
1. Q2: Calculate Edit Distance
2. Q2: Test Edit Distance with given example (leda -> deal)
3. Q3: Sequence Alignment
4. Q3: Test Sequence Alignment with given sequences
5. Run All Test Cases
6. Exit
------------------------------------------------------------

--- Q2: EDIT DISTANCE CALCULATION ---

The edit distance between 'leda' and 'deal' is 3

============================================================
          NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT
============================================================
1. Q2: Calculate Edit Distance
```

2. Q2: Test Edit Distance with given example (leda -> deal)
3. Q3: Sequence Alignment
4. Q3: Test Sequence Alignment with given sequences
5. Run All Test Cases
6. Exit

--- Q2: TESTING WITH GIVEN EXAMPLE ---
The edit distance between 'leda' and 'deal' is 3
Expected: 3 (from manual calculation)

==========
NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT
==========
1. Q2: Calculate Edit Distance
2. Q2: Test Edit Distance with given example (leda -> deal)
3. Q3: Sequence Alignment
4. Q3: Test Sequence Alignment with given sequences
5. Run All Test Cases
6. Exit

--- Q3: SEQUENCE ALIGNMENT ---
Enter sequences (press Enter to use default sequences):

Input Text A: hfddghfgfhgdghdhgfghf
Input Text B: jdgdghfhgffjhhghj

Output:
hfd-dghf-gfhgdghdhgfghf
-jdgdghfhgf--fjh-h--ghj

==========
NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT
==========
1. Q2: Calculate Edit Distance
2. Q2: Test Edit Distance with given example (leda -> deal)
3. Q3: Sequence Alignment
4. Q3: Test Sequence Alignment with given sequences
5. Run All Test Cases
6. Exit

--- Q3: TESTING WITH GIVEN SEQUENCES ---

Input Text A: AGGCTATCACCTGACCTCCAGGCCGATGCCC
Input Text B: TAGCTATCACGACCGCGGTCGATTTGCCCGAC

Output:
-AGGCTATCACCTGACCTCCAGGCCGA--TG-CC--C
TA-GCTATCA-C-GACC-GC-GGTCGATTTGCCCGAC

```
============================================================
          NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT
============================================================
1. Q2: Calculate Edit Distance
2. Q2: Test Edit Distance with given example (leda -> deal)
3. Q3: Sequence Alignment
4. Q3: Test Sequence Alignment with given sequences
5. Run All Test Cases
6. Exit
------------------------------------------------------------

--- RUNNING ALL TEST CASES ---

=== TEST CASES FOR Q2: EDIT DISTANCE ===
Test 1: 'leda' -> 'deal' | Expected: 3, Got: 3 | PASS
Test 2: 'drive' -> 'brief' | Expected: 3, Got: 3 | PASS
Test 3: 'drive' -> 'divers' | Expected: 3, Got: 3 | PASS
Test 4: 'kitten' -> 'sitting' | Expected: 3, Got: 3 | PASS
Test 5: 'abc' -> 'def' | Expected: 3, Got: 3 | PASS
Test 6: '' -> 'hello' | Expected: 5, Got: 5 | PASS
Test 7: 'world' -> '' | Expected: 5, Got: 5 | PASS
Test 8: 'same' -> 'same' | Expected: 0, Got: 0 | PASS
Test 9: 'a' -> 'b' | Expected: 1, Got: 1 | PASS
Test 10: 'insert' -> 'in' | Expected: 4, Got: 4 | PASS

=== TEST CASES FOR Q3: SEQUENCE ALIGNMENT ===

Test 1: Given DNA sequences from problem
Sequence 1: AGGCTATCACCTGACCTCCAGGCCGATGCCC
Sequence 2: TAGCTATCACGACCGCGGTCGATTTGCCCGAC
Aligned 1:  -AGGCTATCACCTGACCTCCAGGCCGA--TG-CC--C
Aligned 2:  TA-GCTATCA-C-GACC-GC-GGTCGATTTGCCCGAC

Test 2: Simple DNA sequence
Sequence 1: ACGT
Sequence 2: AGT
Aligned 1:  ACGT
```

Aligned 2: A-GT

Test 3: Identical sequences
Sequence 1: ATCG
Sequence 2: ATCG
Aligned 1: ATCG
Aligned 2: ATCG

Test 4: Completely different sequences
Sequence 1: AAAA
Sequence 2: TTTT
Aligned 1: AAAA
Aligned 2: TTTT

Test 5: Subsequence alignment
Sequence 1: ATCGATCG
Sequence 2: ATC
Aligned 1: ATCGATCG
Aligned 2: ----ATC-

==
NLP LAB 2: EDIT DISTANCE & SEQUENCE ALIGNMENT
==
1. Q2: Calculate Edit Distance
2. Q2: Test Edit Distance with given example (leda -> deal)
3. Q3: Sequence Alignment
4. Q3: Test Sequence Alignment with given sequences
5. Run All Test Cases
6. Exit

Exiting program. Thank you!