**3. Investigate the Minimum Edit Distance (MED) algorithm and its application in string comparison and the goal is to understand how the algorithm efficiently computes the minimum number of edit operations required to transform one string into another. ● Test the algorithm on strings with different type of variations (e.g., typos, substitutions, insertions, deletions)**

**● Evaluate its adaptability to different types of input variations**

**Program:**

def min\_edit\_distance(str1, str2):

    m = len(str1)

    n = len(str2)

    # Create a DP table to store results of subproblems

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

    # Initialize the base cases

    for i in range(m + 1):

        dp[i][0] = i  # Deletion cost

    for j in range(n + 1):

        dp[0][j] = j  # Insertion cost

    # Fill the DP table

    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 operation needed

            else:

                dp[i][j] = 1 + min(

                    dp[i - 1][j],  # Deletion

                    dp[i][j - 1],  # Insertion

                    dp[i - 1][j - 1]  # Substitution

                )

    # The final result is in dp[m][n]

    return dp[m][n]

# Test cases

test\_cases = [

    ("kitten", "sitting"),  # Substitutions and insertions

    ("intention", "execution"),  # Substitutions and deletions

    ("flaw", "lawn"),  # Substitutions

    ("apple", "aple"),  # Deletion

    ("book", "books"),  # Insertion

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

    ("", "abc"),  # All insertions

    ("abc", "")  # All deletions

]

# Evaluate MED for each test case

for str1, str2 in test\_cases:

    distance = min\_edit\_distance(str1, str2)

    print(f"MED between '{str1}' and '{str2}': {distance}")

**Output:**

MED between 'kitten' and 'sitting': 3

MED between 'intention' and 'execution': 5

MED between 'flaw' and 'lawn': 2

MED between 'apple' and 'aple': 1

MED between 'book' and 'books': 1

MED between 'abc' and 'def': 3

MED between '' and 'abc': 3

MED between 'abc' and '': 3

**Explanation:**

The **Minimum Edit Distance (MED)** algorithm is a dynamic programming approach used to measure the similarity between two strings. It calculates the minimum number of operations required to transform one string into another.

1. **Substitutions and Insertions**:
   * "kitten" → "sitting": Replace 'k' with 's', replace 'e' with 'i', and insert 'g'.
   * MED = 3.
2. **Substitutions and Deletions**:
   * "intention" → "execution": Replace 'i' with 'e', replace 'n' with 'x', delete 'n'.
   * MED = 5.
3. **Substitutions**:
   * "flaw" → "lawn": Replace 'f' with 'l', replace 'w' with 'n'.
   * MED = 2.
4. **Deletion**:
   * "apple" → "aple": Delete 'p'.
   * MED = 1.
5. **Insertion**:
   * "book" → "books": Insert 's'.
   * MED = 1.
6. **All Substitutions**:
   * "abc" → "def": Replace all characters.
   * MED = 3.
7. **All Insertions**:
   * "" → "abc": Insert all characters.
   * MED = 3.
8. **All Deletions**:
   * "abc" → "": Delete all characters.
   * MED = 3.
9. **Write a program to implement top-down and bottom-up parser using appropriate context free grammar.**

**Program:**

import nltk

from nltk import CFG

# Define a simple Context-Free Grammar (CFG)

grammar = CFG.fromstring("""

    S -> NP VP

    NP -> Det N | N

    VP -> V NP | V

    Det -> 'the' | 'a'

    N -> 'cat' | 'dog'

    V -> 'chased' | 'barked'

""")

# Create Top-Down (Recursive Descent) and Bottom-Up (Chart) parsers

top\_down\_parser = nltk.RecursiveDescentParser(grammar)

bottom\_up\_parser = nltk.ChartParser(grammar)

# Input sentence

sentence = "the cat chased a dog".split()

# Top-Down Parsing

print("Top-Down Parsing Results:")

for tree in top\_down\_parser.parse(sentence):

    print(tree)

# Bottom-Up Parsing

print("\nBottom-Up Parsing Results:")

for tree in bottom\_up\_parser.parse(sentence):

    print(tree)

**Output:**

Top-Down Parsing Results:

(S (NP (Det the) (N cat)) (VP (V chased) (NP (Det a) (N dog))))

Bottom-Up Parsing Results:

(S (NP (Det the) (N cat)) (VP (V chased) (NP (Det a) (N dog))))