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, n = len(str1), len(str2)

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

for i in range(m + 1):

for j in range(n + 1):

if i == 0:

dp[i][j] = j # Insertions

elif j == 0:

dp[i][j] = i # Deletions

elif str1[i - 1] == str2[j - 1]:

dp[i][j] = dp[i - 1][j - 1] # No change

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]

# Test Cases

test\_cases = [

("kitten", "sitting"), # Substitutions, Insertions

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

("intention", "execution"), # Mixed operations

("abcdef", "azced"), # Substitutions & Deletions

("hello", "helo"), # Deletion

("abc", ""), # Deletion to empty string

("", "abc"), # Insertion from empty string

]

# Run tests

for str1, str2 in test\_cases:

print(f"MED('{str1}', '{str2}') = {min\_edit\_distance(str1, str2)}")