Edit Distance C++

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
int main() {
  string s1 = "cat";
  string s2 = "cut";
  int m = s1.length();
  int n = s2.length();
  // Initialize the 2D array with dimensions (m+1) x
  int dp[m + 1][n + 1] = \{0\};
  // Fill the dp array
  for (int i = 0; i \le m; i++) {
     for (int j = 0; j \le n; j++) {
        if (i == 0) {
          dp[i][j] = j; // If s1 is empty, insert all
characters of s2
        else if (j == 0) {
          dp[i][j] = i; // If s2 is empty, remove all
characters of s1
        } else {
          int f1 = 1 + dp[i - 1][j - 1]; // Replace
          int f2 = 1 + dp[i - 1][j]; // Delete
          int f3 = 1 + dp[i][j - 1];
                                     // Insert
          dp[i][j] = min(\{f1, f2, f3\});
     }
  }
  cout << dp[m][n] << endl; // Output the result
  return 0;
}
```

Dry Run of the Program

Let's go through the dry run of the code with the input strings s1 = "cat" and s2 = "cut".

Input:

- s1 = "cat"
- s2 = "cut"
- m = 3 (Length of s1)
- n = 3 (Length of s2)

Step 1: Initialize the dp array

We create a 2D DP table with dimensions (m+1) x (n+1), which is a 4x4 table since m = 3 and n = 3.

int $dp[4][4] = \{0\};$

Step 2: Fill the dp table

Now, let's fill the table using the given conditions.

- 1. When i = 0 (Empty string s1):
 - o dp[0][0] = 0 (Both strings are empty)
 - dp[0][1] = 1 (Insert 1 character 'c' from s2)
 - o dp[0][2] = 2 (Insert 2 characters 'cu' from s2)
 - o dp[0][3] = 3 (Insert 3 characters 'cut' from s2)

The first row looks like this:

$$dp[0] = \{0, 1, 2, 3\}$$

- 2. When j = 0 (Empty string s2):
 - dp[1][0] = 1 (Remove 1 character 'c' from s1)
 - o dp[2][0] = 2 (Remove 2 characters 'ca' from s1)
 - o dp[3][0] = 3 (Remove 3 characters 'cat' from s1)

The first column looks like this:

$$dp[1] = \{1, 0, 0, 0\}$$

 $dp[2] = \{2, 0, 0, 0\}$
 $dp[3] = \{3, 0, 0, 0\}$

- 3. **When i = 1 and j = 1** (comparing 'c' and 'c'):
 - Since s1[0] == s2[0], no operation is required.
 - \circ dp[1][1] = dp[0][0] = 0

After this, the table looks like this:

$$dp[1] = \{1, 0, 0, 0\}$$

- 4. When i = 1 and j = 2 (comparing 'c' and 'u'):
 - We need to perform an **insert** operation.
 - o dp[1][2] = 1 + min(dp[0][2], dp[1] [1], dp[0][1]) = 1 + min(2, 0, 1) = 1 + 1 = 2

After this step, the table looks like:

$$dp[1] = \{1, 0, 2, 0\}$$

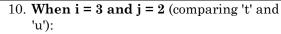
- 5. When i = 1 and j = 3 (comparing 'c' and 't'):
 - We need to perform an **insert** operation.
 - o dp[1][3] = 1 + min(dp[0][3], dp[1] [2], dp[0][2]) = 1 + min(3, 2, 2) = 1 + 2 = 3

After this step, the table looks like:

$$dp[1] = \{1, 0, 2, 3\}$$

Continue filling the rest of the table similarly:

- 6. When i = 2 and j = 1 (comparing 'a' and 'c'):
 - We need to perform a **delete** operation.
 - o dp[2][1] = 1 + min(dp[1][1], dp[2] [0], dp[1][0]) = 1 + min(0, 2, 1) = 1 + 0 = 1
- 7. **When i = 2 and j = 2** (comparing 'a' and 'u'):
 - We need to perform a **replace** operation.
 - o dp[2][2] = 1 + min(dp[1][2], dp[2] [1], dp[1][1]) = 1 + min(2, 1, 0) = 1 + 0 = 1
- 8. **When i = 2 and j = 3** (comparing 'a' and 't'):
 - We need to perform an **insert** operation.
 - o dp[2][3] = 1 + min(dp[1][3], dp[2] [2], dp[1][2]) = 1 + min(3, 1, 2) = 1 + 1 = 2
- 9. **When i = 3 and j = 1** (comparing 't' and 'c'):
 - We need to perform a **delete** operation.
 - o dp[3][1] = 1 + min(dp[2][1], dp[3] [0], dp[2][0]) = 1 + min(1, 3, 2) = 1 + 1 = 2



- We need to perform a **replace** operation.
- o dp[3][2] = 1 + min(dp[2][2], dp[3] [1], dp[2][1]) = 1 + min(1, 2, 1) = 1 + 1 = 2
- 11. **When i = 3 and j = 3** (comparing 't' and 't'):
 - Since s1[2] == s2[2], no operation is required.
 - o dp[3][3] = dp[2][2] = 1

Final DP Table:

$$dp[0] = \{0, 1, 2, 3\}$$

$$dp[1] = \{1, 0, 2, 3\}$$

$$dp[2] = \{2, 1, 1, 2\}$$

$$dp[3] = \{3, 2, 2, 1\}$$

Final Output:

The value at dp[m][n] is dp[3][3] = 3.

So, the minimum number of operations (insertions, deletions, or replacements) required to convert "cat" to "cut" is $\bf 3$.

Output:-