Dynamic Programming | Set 5 (Edit Distance)

Given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of edits (operations) required to convert ‘str1′ into ‘str2′.

a. Insert
b. Remove
c. Replace

All of the above operations are of equal cost.

Examples:

```
Input:   str1 = "geek", str2 = "gesek"
Output:  1
We can convert str1 into str2 by inserting a 's'.

Input:   str1 = "cat", str2 = "cut"
Output:  1
We can convert str1 into str2 by replacing 'a' with 'u'.

Input:   str1 = "sunday", str2 = "saturday"
Output:  3
Last three and first characters are same.  We basically
need to convert "un" to "atur".  This can be done using
below three operations.
Replace 'n' with 'r', insert t, insert a
```

What are the subproblems in this case?

The idea is process all characters one by one staring from either from left or right sides of both strings.
Let we traverse from right corner, there are two possibilities for every pair of character being traversed.

```
m: Length of str1 (first string)
n: Length of str2 (second string)
```

1. If last characters of two strings are same, nothing much to do. Ignore last characters and get count for remaining strings. So we recur for lengths m-1 and n-1.
2. Else (If last characters are not same), we consider all operations on ‘str1′, consider all three operations on last character of first string, recursively compute minimum cost for all three operations and take minimum of three values.
    a. Insert: Recur for m and n-1
    b. Remove: Recur for m-1 and n
    c. Replace: Recur for m-1 and n-1

Below is C++ implementation of above Naive recursive solution.

```
// A Naive recursive C++ program to find minimum number
// operations to convert str1 to str2
#include<bits/stdc++.h>
using namespace std;

// Utility function to find minimum of three numbers
int min(int x, int y, int z)
{
   return min(min(x, y), z);
}

int editDist(string str1 , string str2 , int m ,int n)
{
    // If first string is empty, the only option is to
    // insert all characters of second string into first
    if (m == 0) return n;

    // If second string is empty, the only option is to
    // remove all characters of first string
    if (n == 0) return m;

    // If last characters of two strings are same, nothing
    // much to do. Ignore last characters and get count for
    // remaining strings.
    if (str1[m-1] == str2[n-1])
        return editDist(str1, str2, m-1, n-1);

    // If last characters are not same, consider all three
    // operations on last character of first string, recursively
    // compute minimum cost for all three operations and take
    // minimum of three values.
    return 1 + min ( editDist(str1,  str2, m, n-1),    // Insert
                     editDist(str1,  str2, m-1, n),   // Remove
                     editDist(str1,  str2, m-1, n-1) // Replace
                   );
}

// Driver program
int main()
{
    // your code goes here
    string str1 = "sunday";
    string str2 = "saturday";

    cout << editDist( str1 , str2 , str1.length(), str2.length());

    return 0;
}
```

Run on IDE

Output:

```
3
```

The time complexity of above solution is exponential. In worst case, we may end up doing $O(3^m)$ operations. The worst case happens when none of characters of two strings match. Below is a recursive call diagram for worst case.

We can see that many subproblems are solved again and again, for example eD(2,2) is called three times. Since same suproblems are called again, this problem has Overlapping Subprolems property. So Edit Distance problem has both properties (see this and this) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array that stores results of subpriblems.

```
// A Dynamic Programming based C++ program to find minimum
// number operations to convert str1 to str2
#include<bits/stdc++.h>
using namespace std;
```

```
// Utility function to find minimum of three numbers
int min(int x, int y, int z)
{
    return min(min(x, y), z);
}

int editDistDP(string str1, string str2, int m, int n)
{
    // Create a table to store results of subproblems
    int dp[m+1][n+1];

    // Fill d[][] in bottom up manner
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            // If first string is empty, only option is to
            // isnert all characters of second string
            if (i==0)
                dp[i][j] = j;  // Min. operations = j

            // If second string is empty, only option is to
            // remove all characters of second string
            else if (j==0)
                dp[i][j] = i; // Min. operations = i

            // If last characters are same, ignore last char
            // and recur for remaining string
            else if (str1[i-1] == str2[j-1])
                dp[i][j] = dp[i-1][j-1];

            // If last character are different, consider all
            // possibilities and find minimum
            else
                dp[i][j] = 1 + min(dp[i][j-1],  // Insert
                                   dp[i-1][j],  // Remove
                                   dp[i-1][j-1]); // Replace
        }
    }

    return dp[m][n];
}

// Driver program
int main()
{
    // your code goes here
    string str1 = "sunday";
    string str2 = "saturday";

    cout << editDistDP(str1, str2, str1.length(), str2.length());

    return 0;
}
```

Run on IDE

Output:

```
3
```

Time Complexity: O(m x n)

Auxiliary Space: O(m x n)

Applications: There are many practical applications of edit distance algorithm, refer Lucene API for sample. Another example, display all the words in a dictionary that are near proximity to a given word\incorrectly spelled word.

Thanks to Vivek Kumar for suggesting above updates.