https://discuss.leetcode.com/topic/50687/simple-dp-solution-4ms-beats-52-c
https://leetcode.com/problems/interleaving-string/#/description

Problem Statement

Given *s1*, *s2*, *s3*, find whether *s3* is formed by the interleaving of *s1* and *s2*.
For example,
Given:
s1 = "aabcc",
s2 = "dbbca",
When *s3* = "aadbbcbcac", return true.
When *s3* = "aadbbbaccc", return false.

As this editorial is already available, I thought to give another approach which is easier to understand. Therefore, I am skipping brute force and other approaches and explaining only new approach.

Approach # New : DP solution

	S1 ---------->				
S2 | | |		0	a	a	b
	0	1[0,0]	[0,1]	[0,2]	[0,3]
	a	[1,0]	[1,1]	[1,2]	[1,3]
	b	[2,0]	[2,1]	[2,2]	[2,3]

Motivation

Let there be a matrix where two side represents strings s1 and s2 [see above fig].
First row denotes characters of string s1 and first column denotes characters of string s2.

If we do interleaving of two strings, at any point we have to choose from

1. String s1 or from
2. String s2

The above matrix denotes all possible interleavings in form of paths from cell [0,0] to rightmost bottom cell.

For any string to be interleaving of two other string, there must be a path in 2-D matrix,
for e.g.
For reaching to cell [1,1] it denotes choosing path from
[0,0] - [0,1] - [1,1] i.e. NULL - a(of string s1) - a(of string s2)
There can be other path also to come to [1,1] which is
[0,0] - [1,0] - [1,1] i.e. NULL - a(of string s2) - a(of string s1)

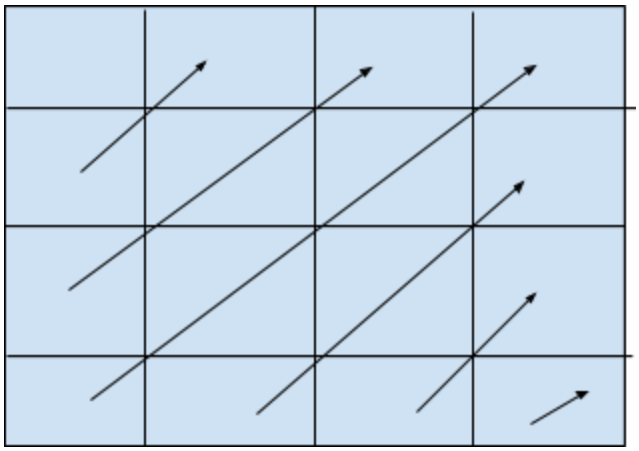
So for reaching to particular cell, there are two paths available

1. From above cell
2. From left cell

So, motivation is from the fact that if there is an interleaving of two string then there must exist a path from [0,0] to rightmost bottom cell by which interleaving have occurred because while picking characters from two strings we are taking a path (either moving right or bottom) in this above 2-D matrix.

Algorithm

1. We start with dp[0][0] as 1 i.e NULL string.
2. we move along second principal diagonal manner i.e. dp[1][0] to dp[0][1]



   then dp[2][0] to dp[1][1] to dp[0][2] and increase s3 string pointer
3. So, for reaching current cell, there are two paths only i.e. from above cell and from left cell
4. If above cell has value 1 and s3 character is equal to s2 string character at that cell or left cell has value 1 and s3 character is equal to s1 character at that cell, then we put 1 in the cell.
5. Otherwise, value is made 0.
6. We return true if rightmost bottom element is 1.

Complexity Analysis

Space Complexity O(n^2) - for 2-d matrix
Time Complexity O(n^2) - as we are traversing 2-d matrix once.

Code

```
public:
    bool isInterleave(string s1, string s2, string s3) {
        // base cases
       if(s1.size()+s2.size()!=s3.size())return false;
       if(s1.size()==0 && s2==s3)return true;
       else if(s1.size()==0 && s2!=s3)return false;
       else if(s2.size()==0 && s1==s3)return true;
       else if(s2.size()==0 && s1!=s3)return false;

       s1.insert(0,"0");
       s2.insert(0,"0");
       int l1=s1.length();
       int l2=s2.length();
       int l3=s3.length();
       vector<vector<int>>dp(l2+10,vector<int>(l1+10,0));
       dp[0][0]=1;
          // Pointers i1 for s1 i2 for s2 i3 for s3
       int i3=0;
       int i1=1,j1=0;
       while(1)
       {
          int i=i1,j=j1;
          // going in second principal diagonal form
          while(i>=0 && j<l1)
          {
           // if reached to current cell from above cell
              if(i-1>=0 && dp[i-1][j]==1)
                  if(s3[i3]==s2[i])
                      dp[i][j]=1;
          // if reached the current cell from left cell
              if(j-1>=0 && dp[i][j-1]==1)
                  if(s3[i3]==s1[j])
                      dp[i][j]=1;
              i--,j++;
          }
          // reaced end of both strings
          if(i1==l2-1 && j1==l1-1)break;
          i3++;
          if(i1<l2-1)
              i1++;
```

```
            else j1++;
        }
        return dp[l2-1][l1-1];
    }
};
```