D.P.

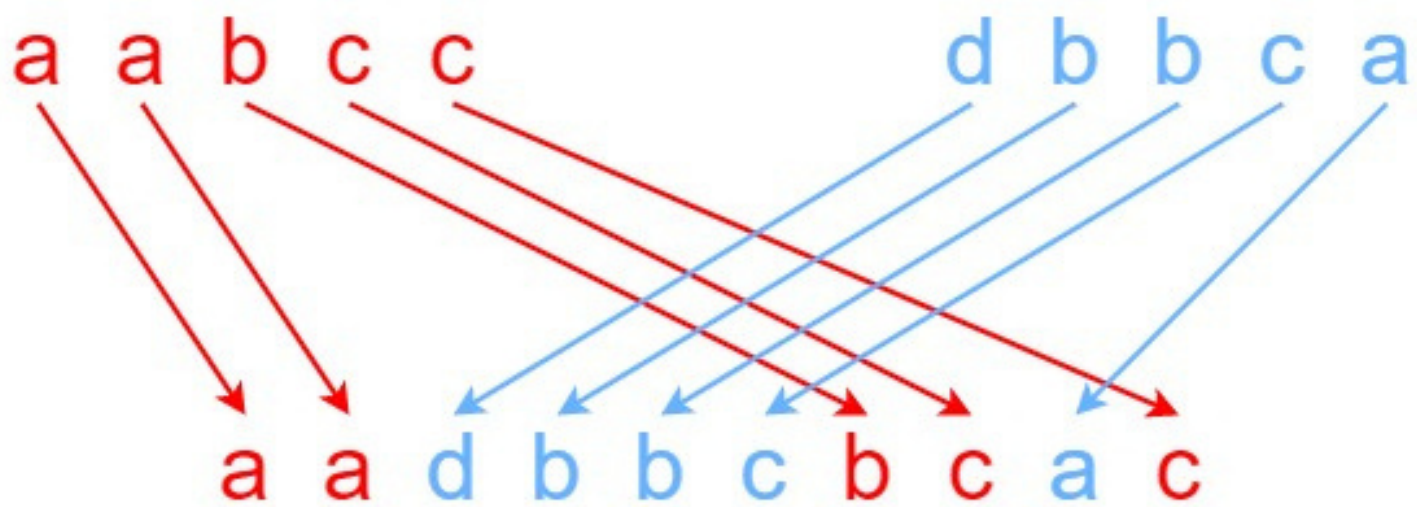
Interleaving String

Today's Leetcode Problem

Mayank

30% Accuracy

Given strings s1, s2, and s3, find whether s3 is formed by an interleaving of s1 and s2



i/p

```
s1 = "aabcc", s2 = "dbbca",
s3 = "aadbbcbcac"
```

o/p

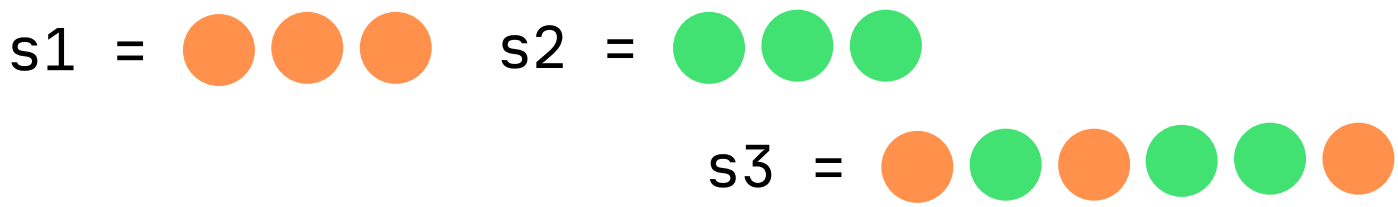
```
true
```

constraints:

- 0 <= s1.length, s2.length <= 105
- 0 <= s3.length <= 200

How to approach ?

let's say you have s1 = "aab" s2 = "bbc" s3 = "ababbc"



In the above diagram I have replaced all characters of s1 with orange circle & s2 with green circles

s3 contains circles from both s1 & s2

Now, how you would you check whether s3 can be formed using s1 & s2 ???

How to approach ?

s1 = 🟠🟠🟠 s2 = 🟢🟢🟢

s3 = 🟠🟢🟠🟢🟢🟠

Let's start from 0th index of s1, s2 & s3

0th circle in s3 is 🟠 which comes from 0th circle of s1 as s2 has a different one (🟢)

Now, you are remaining to check below circles-

s1 = 🟠🟠 s2 = 🟢🟢🟢 s3 = 🟢🟠🟢🟢🟠

see we removed 1 circle from s1 & s3 as they matched so now we need a lesser length to check

How to approach ?

s1 =  s2 = 

s3 = 

Let's say we have a function solve(s1,s2,s3,i,j,k) which return true if s3 can be made from s1 & s2

So what we did in our previous slide was

```
if(s1[i] == s3[k]) return solve(s1,s2,s3,i+1,j,k+1)
```

If in place of s1, s2's jth char matched with s3's kth char

```
if(s2[j] == s3[k]) return solve(s1,s2,s3,i,j+1,k+1)
```

I hope you are able to understand the flow (trying my best)

How to approach ?

s1 = 

s2 = 

s3 = 

Consider this example

Now s1[0],s2[0] both are orange, so s3[0] which is also orange, can you guarantee from whose orange circle (s1 or s2) s3[0] is made ?

So here you got a choice

Either match s1[0] with s3[0]

or

match s2[0] with s3[0]

So for every character in s1 & s2 you have a choice to make wrt s3's character

Recursion

Till now we saw a recursive function

I hope you atleast got to know why we will use Recursion here

Already told you about 60% of approach, rest is on you

Showing you a Memoized DP solution

ENJOY...

```cpp
int dp[105][105];


bool solve(string& s1,string& s2,int i,int j,string& s3)
{

    int n1 = s1.length();
    int n2 = s2.length();
    int n3 = s3.length();

    int k = i + j;
    if(i >= n1 && j >= n2 && k >= n3) {
        return 1;
    }

    if(i >= n1 && s3[k] != s2[j]
       || j >= n2 && s3[k] != s1[i]) {
        return 0;
    }

    bool ans = 0;

    if(dp[i][j] == -1) {

        if(s3[k] == s1[i]) {
            ans = ans || solve(s1,s2,i+1,j,s3);
        }
        if(s3[k] == s2[j]) {
            ans = ans || solve(s1,s2,i,j+1,s3);
        }

        dp[i][j] = ans;

    }
    return dp[i][j];

}


class Solution {
public:
    bool isInterleave(string s1, string s2, string s3) {


        memset(dp,-1,sizeof(dp));

        return solve(s1,s2,0,0,s3);


    }
};
```