Expression Matching Problems

Recursion
↓
Memoization
&
Tabulation
&
Space optimization

→ Shortest Common Supersequence | LC 1092 | Hard
→ Distinct Subsequences | LC 115 | Hard
→ Edit Distance | LC 72 | Hard
→ Wildcard Matching | LC 44 | Hard
→ RegEx Matching | LC 10 | Hard
→ Min Operations to make A → B → Min ASCII Delete Sum

Interleaving String
Min Insertion/Deletion

Shortest Common Supersequence

LC 1092, hard

contains both S1 & S2 as a subsequence

↑ Common Supersequences

"archit" "rohini"

"rhi" LCS

"architrohini", "rohiniarchit", "arrcochhiitnit"

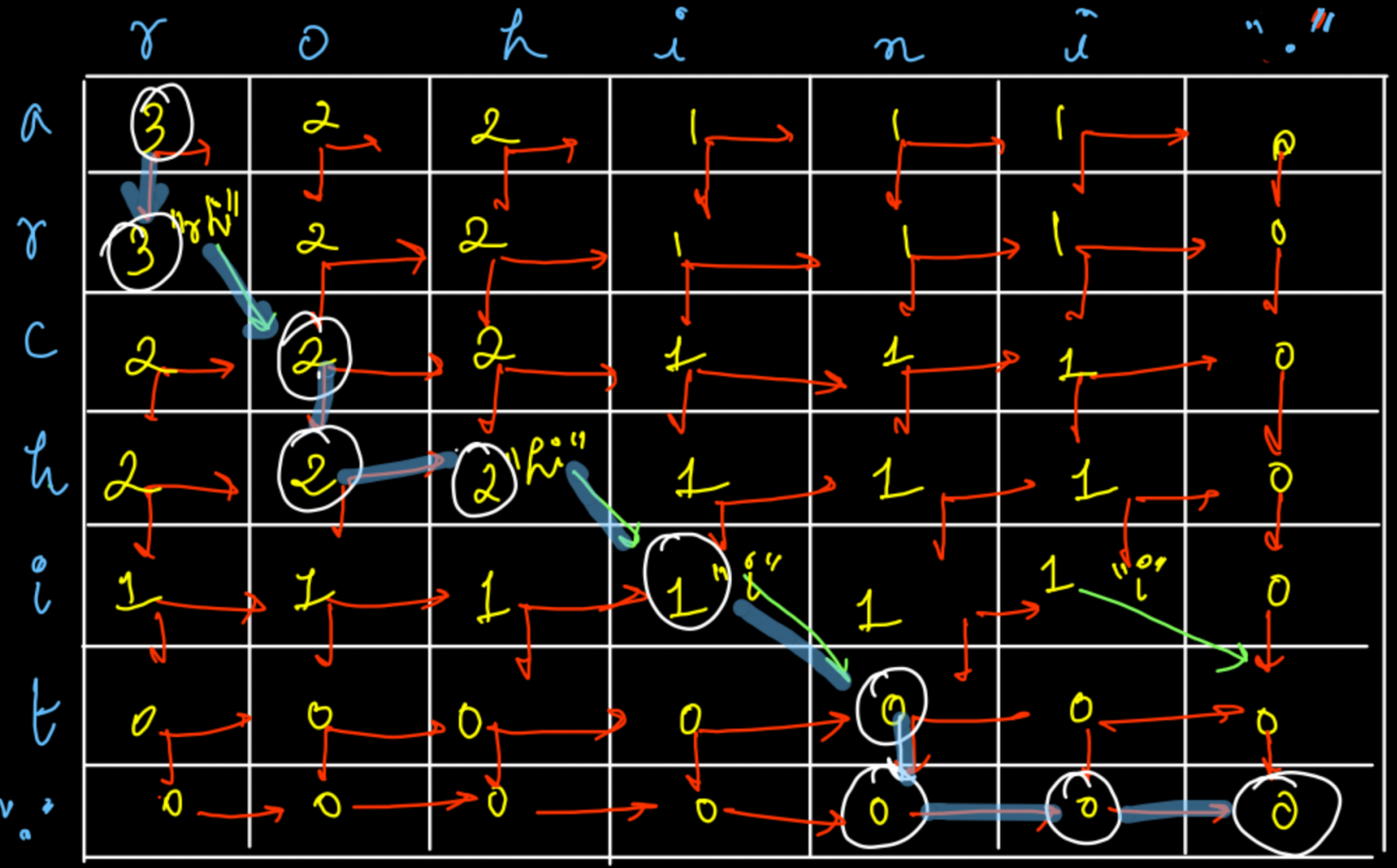
Shortest Common Subsequence → "arcohitni"

↓ There may be multiple SCS

$$SCS = n + m - LCS$$ (length)

LCS table

backtrack(SCS) → yes → in both cells, add character in SCS.
→ no



"archiitni"

SCS

```
String scs = "";

public void backtrack(int i, int j, String s1, String s2, int[][] dp, String ans){
    if(i == s1.length() && j == s2.length()){
        scs = ans;
        return;
    }

    char ch1 = (i < s1.length()) ? s1.charAt(i) : 'A';
    char ch2 = (j < s2.length()) ? s2.charAt(j) : 'B';

    if(ch1 == ch2){
        // Yes - Diagonal
        backtrack(i + 1, j + 1, s1, s2, dp, ans + ch1);
    }
    else if(i + 1 <= s1.length() && dp[i][j] == dp[i + 1][j]){
        // No - Down
        backtrack(i + 1, j, s1, s2, dp, ans + ch1);
    } else {
        // No - Right
        backtrack(i, j + 1, s1, s2, dp, ans + ch2);
    }
}
```

↳ last row & & last col → base case → bottom right corner

DP

SCS → O(m*n + (m+n))

Backtracking (Any one SCS)

```
public String shortestCommonSupersequence(String s1, String s2) {
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];

    for(int i=s1.length()-1; i>=0; i--){
        for(int j=s2.length()-1; j>=0; j--){
            char ch1 = s1.charAt(i);
            char ch2 = s2.charAt(j);

            if(ch1 == ch2)
                dp[i][j] = 1 + dp[i + 1][j + 1];

            else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    // If Length of SCS was asked, SCS = N + M - LCS
    backtrack(0, 0, s1, s2, dp, "");
    return scs;
}
```

in terms of index ← Distinct Subsequences

"b'a'b"g'b"ag"" , "bag"

S1 S2

Find all occurences of s2 in s1 in form of subsequence ★

babgbag

babgbag → b'a'g'

babgbag → b'a'g"

babgbag → b'a"g"'

babgbag → b'a"g"

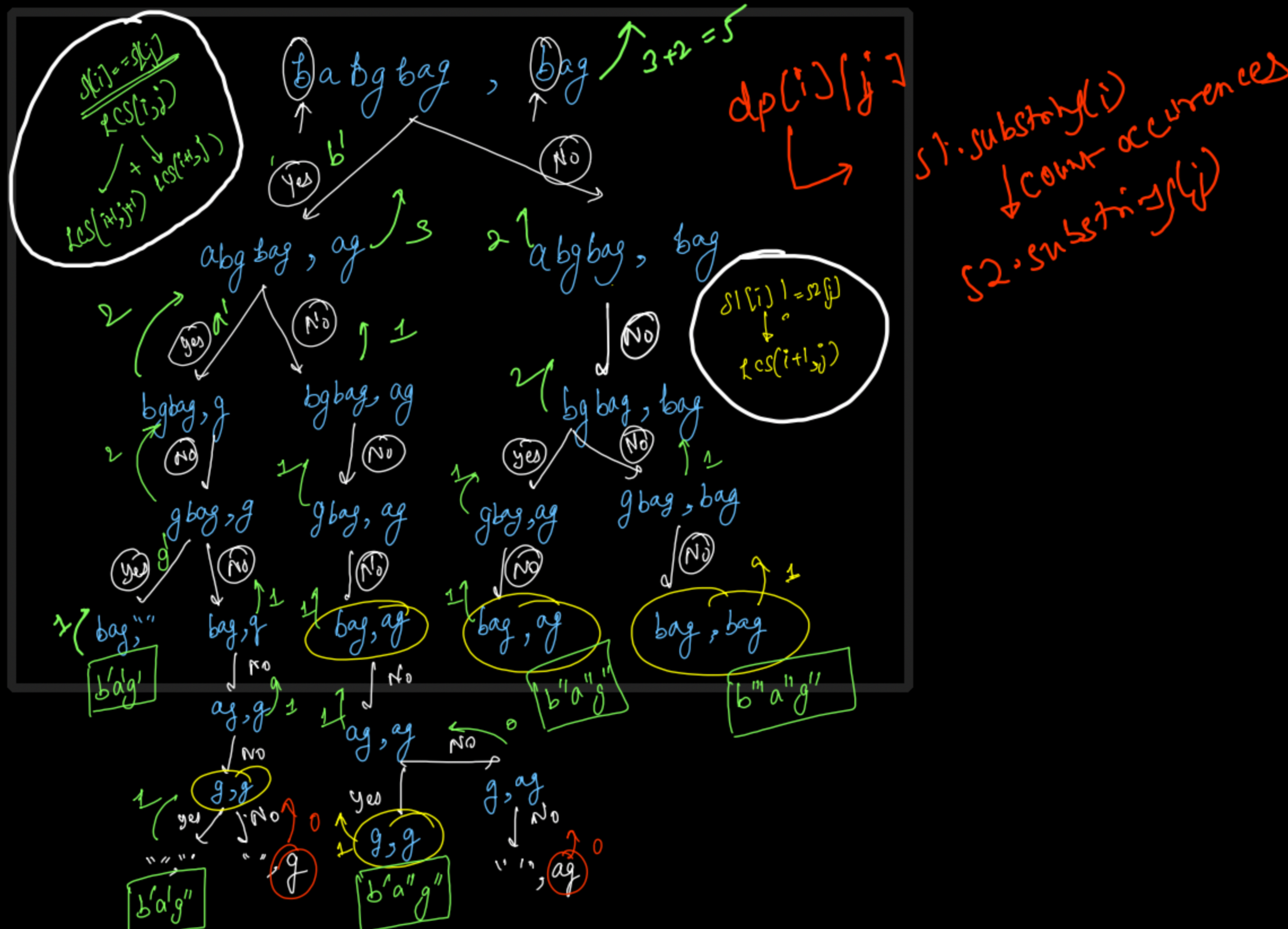
babgbag → b"'a"g"

count = 5

RCS(i,j)

s(i) == s(j) → RCS(i+1, j+1)

s(i) != s(j) → RCS(i, j+1) , RCS(i+1, j)

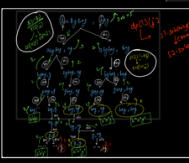

s[i]==s[j]
LCS(i,j)
LCS(i+1,j+1) + LCS(i+1,j)
babgbag , bag
3+2=5
yes b'
No
abgbag , ag
3
2
abgbag , bag
s1[i] != s2[j]
LCS(i+1,j)
bgbag, g
bgbag, ag
bgbag , bag
gbag, g
gbag, ag
gbag, ag
gbag , bag
bag, ""
bag, g
bag, ag
bag , ag
bag , bag
b'a'g'
b"a"g"
b'''a''g''
ag, g
ag, ag
g, g
g, ag
b'a'g"
g
ag
dp[i][j]
s1.substring(i)
count occurences
s2.substring(j)

```java
public int memo(int i, int j, String s1, String s2, int[][] dp){
    if(j == s2.length()) return 1; // Required String (s2) is completely found
    if(i == s1.length()) return 0; // Required is still left, actual string is empty

    if(dp[i][j] != -1) return dp[i][j];

    char actual = s1.charAt(i);
    char required = s2.charAt(j);

    if(actual == required) // both yes and no calls
        return dp[i][j] = (memo(i + 1, j + 1, s1, s2, dp)
            + memo(i + 1, j, s1, s2, dp));

    // if actual != required, only no call
    return dp[i][j] = memo(i + 1, j, s1, s2, dp);
}

public int numDistinct(String s1, String s2) {
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];
    for(int i=0; i<dp.length; i++)
        for(int j=0; j<dp[0].length; j++)
            dp[i][j] = -1;

    return memo(0, 0, s1, s2, dp);
}
```

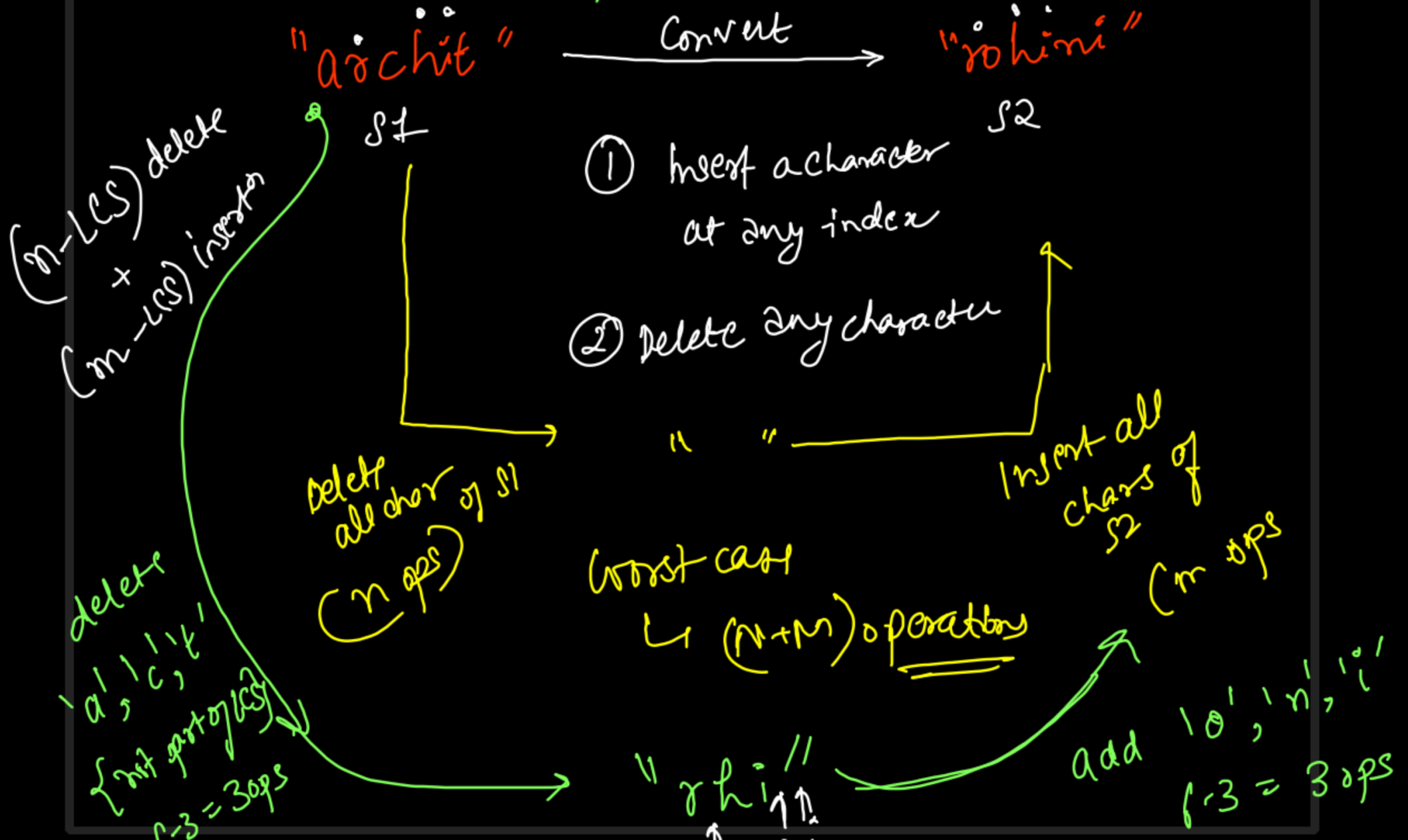
Distinct subsequences

O(N*M) time

O(N*M) 2D



Min Steps to Convert S1 to S2



```java
public int minOperations(String s1, String s2)
{
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];
    for(int i=s1.length()-1; i>=0; i--){
        for(int j=s2.length()-1; j>=0; j--){
            char ch1 = s1.charAt(i);
            char ch2 = s2.charAt(j);

            if(ch1 == ch2)
                dp[i][j] = 1 + dp[i + 1][j + 1];

            else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    int lcs = dp[0][0];
    int deletions = (s1.length() - lcs);
    int insertions = (s2.length() - lcs);
    return deletions + insertions;
}
```

LCS

Time → O(N*M)

Space → O(N*M)