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

leetcode 1092, hard

contains both s1 & s2 as a subsequence

Common Supersequences

"archit" "hi" LCS "ohini"

"architohini", "ohiniarchit", "arrcochhiitnit"

Shortest Common Subsequence → "architni"

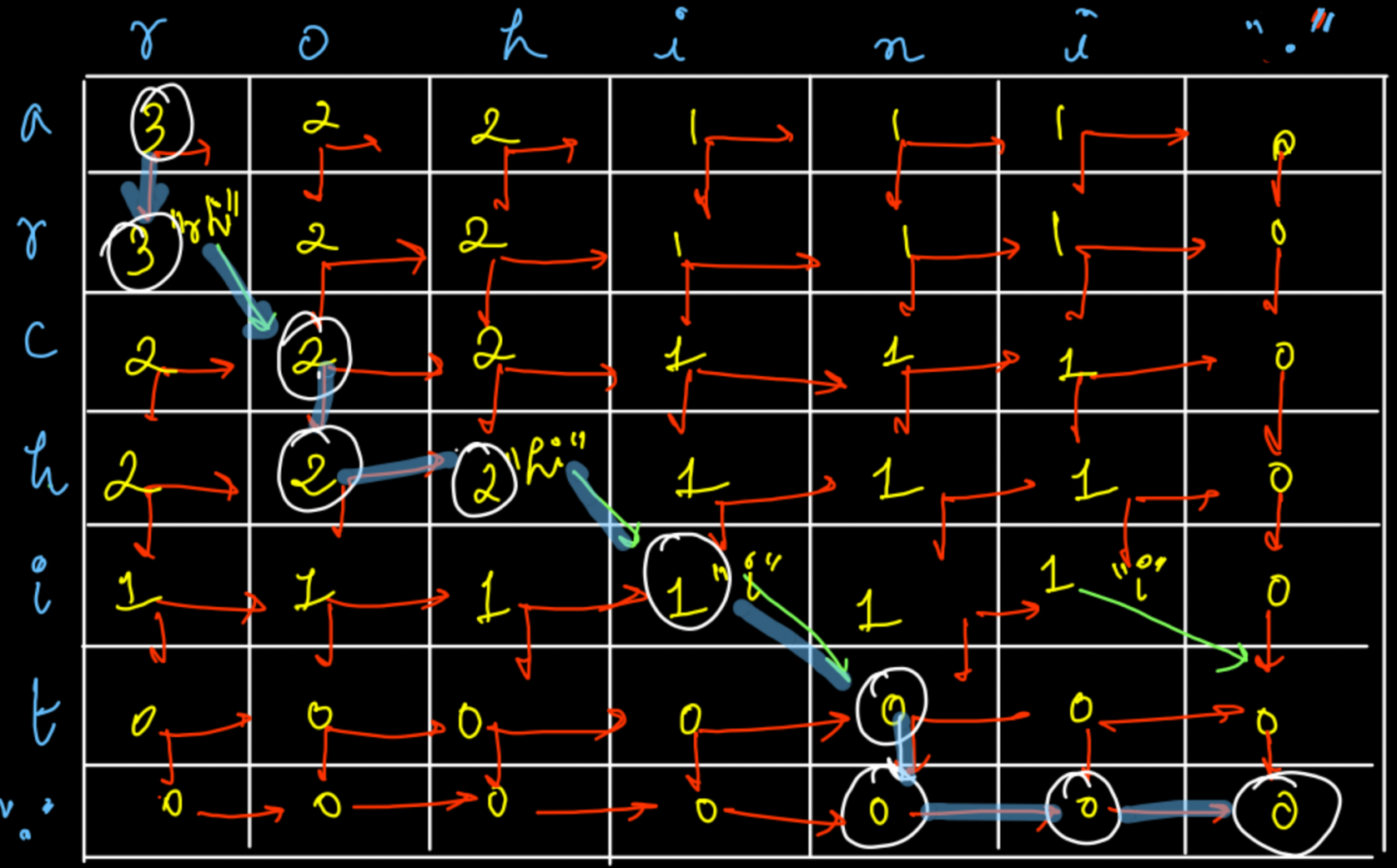
These may be multiple SCS

length

SCS = n + m - LCS

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"

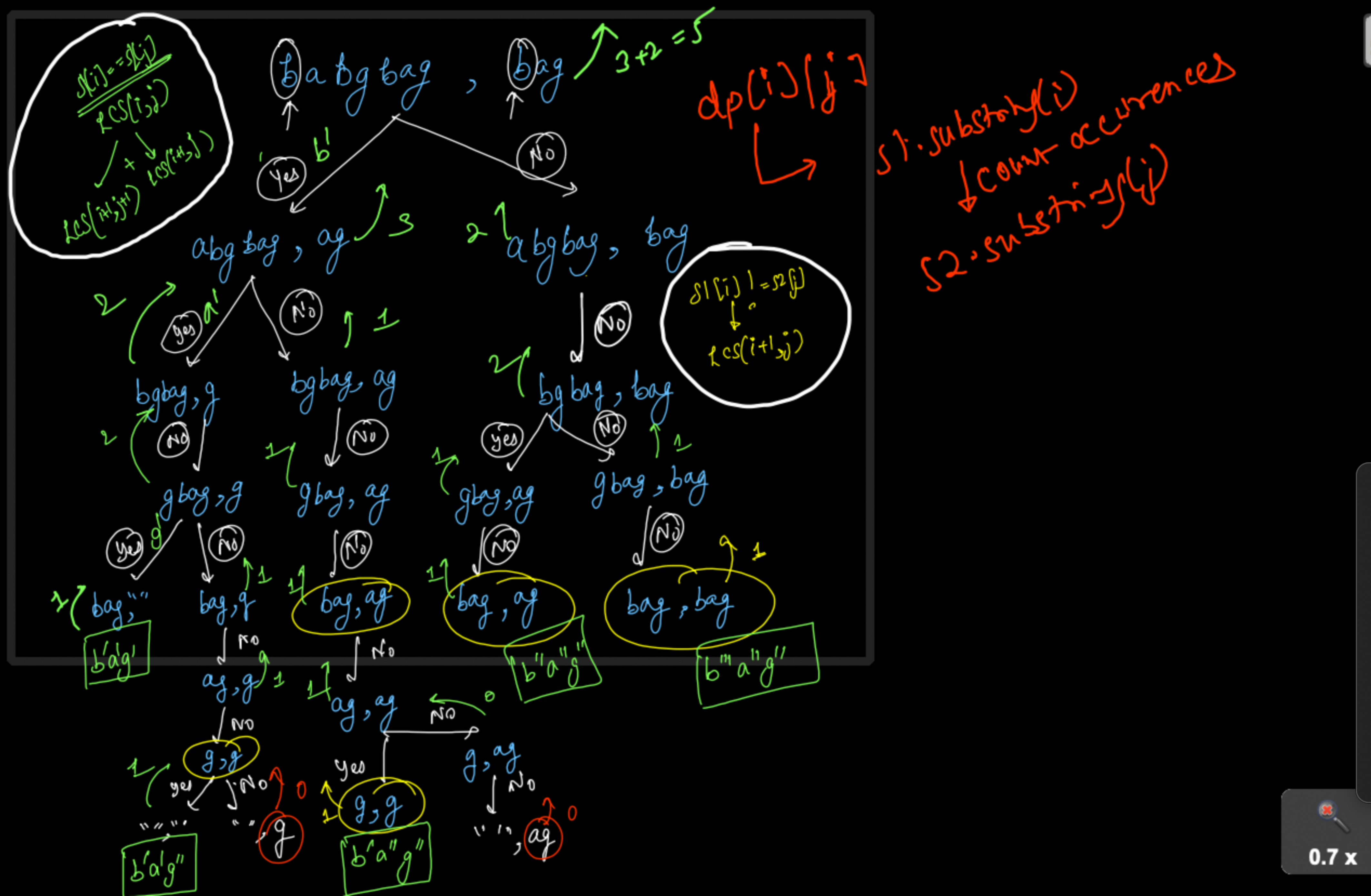
babgbag → b"'a"g"

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)

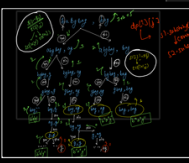
 s[i] == s[j]
LCS(i, j)
LCS(i+1, j+1) + LCS(i+1, j)
babgbag , bag
3+2 = 5
yes
No
abgbag , ag
abgbag , bag
s1[i] != s2[j]
LCS(i+1, j)
bgbag , g
bgbag , ag
bgbag , bag
gbag , g
gbag , ag
gbag , bag
bag , ""
bag , g
bag , ag
bag , bag
ag , g
ag , ag
g , g
g , ag
dp[i][j]
s1 : substring(i)
count occurences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

```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);
}
```

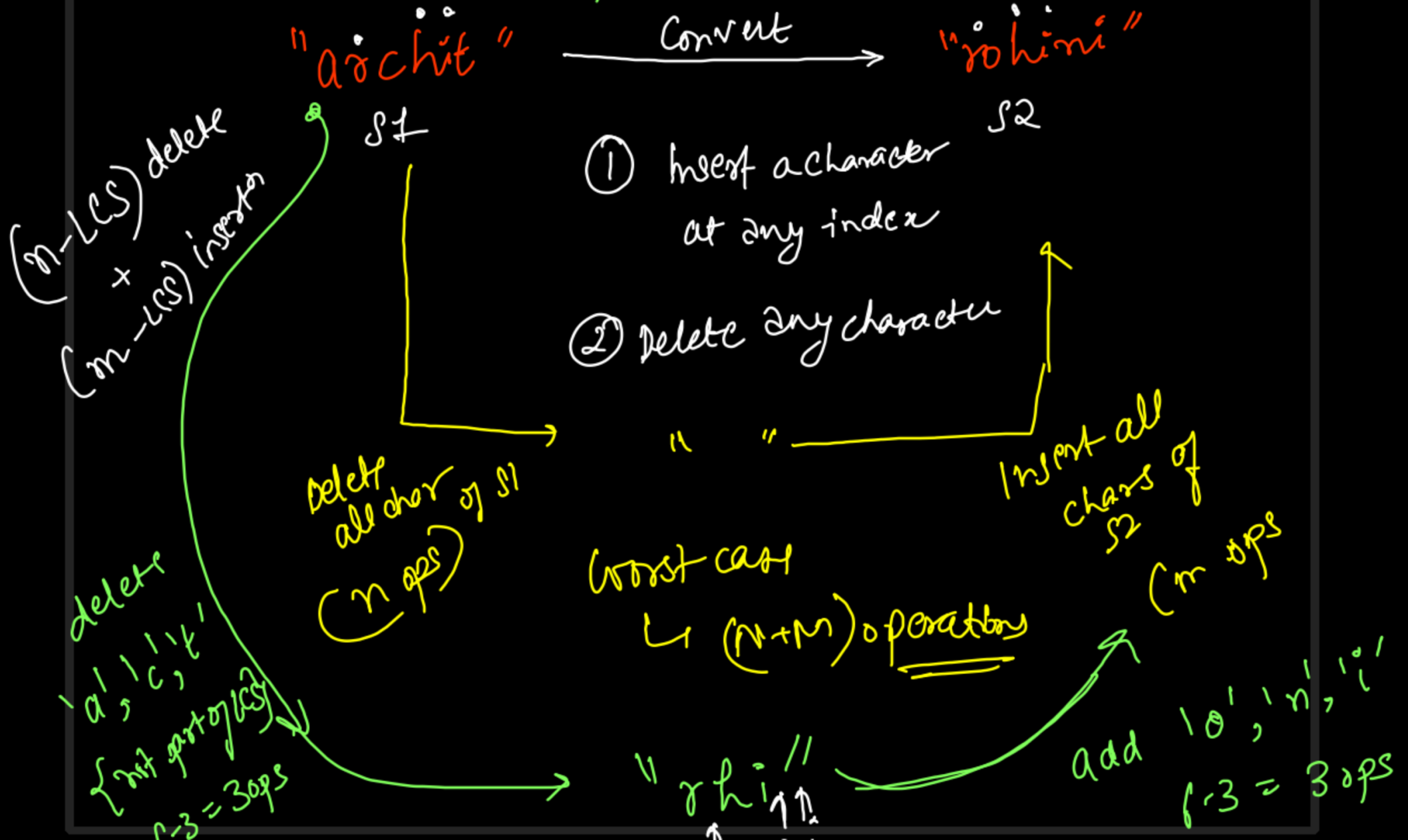
Distinct subsequences

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)

Expression Matching Lecture ②

9:45 pm to 11:45 pm

2 June 2022, Thursday

- → Wildcard Matching (LC 44)
- → Regular Expression Matching (LC 10)
- → Edit Distance (LC 72)
- → Interleaving String (LC 97)
- → min operations to make Strings Equal
  - GFG
  - LC 583
  - LC 712

Wildcard matching

"architagg" — String

"a*h?t*a?g" — Pattern

src, i, j

`i == n && j == m` → return true

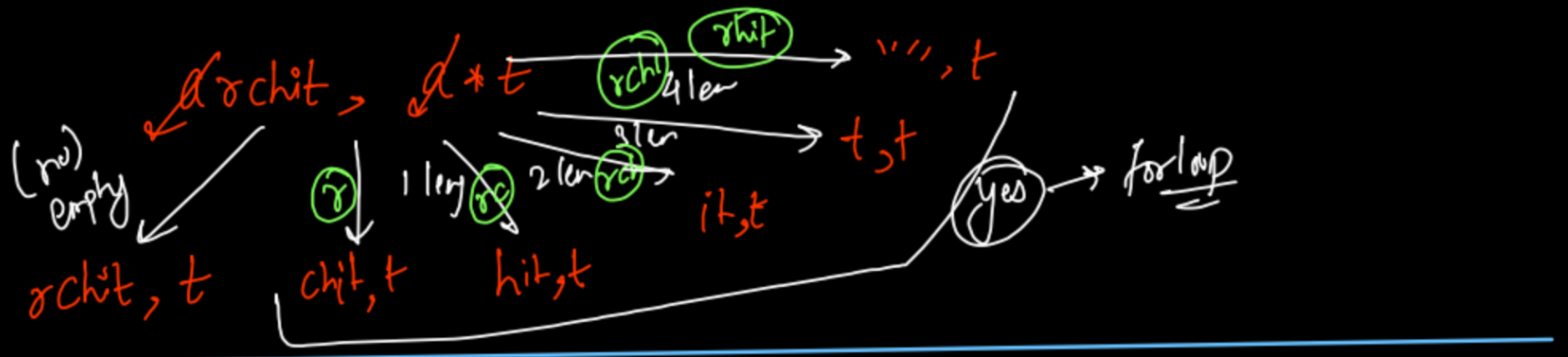
`i == n || j == m` → return false

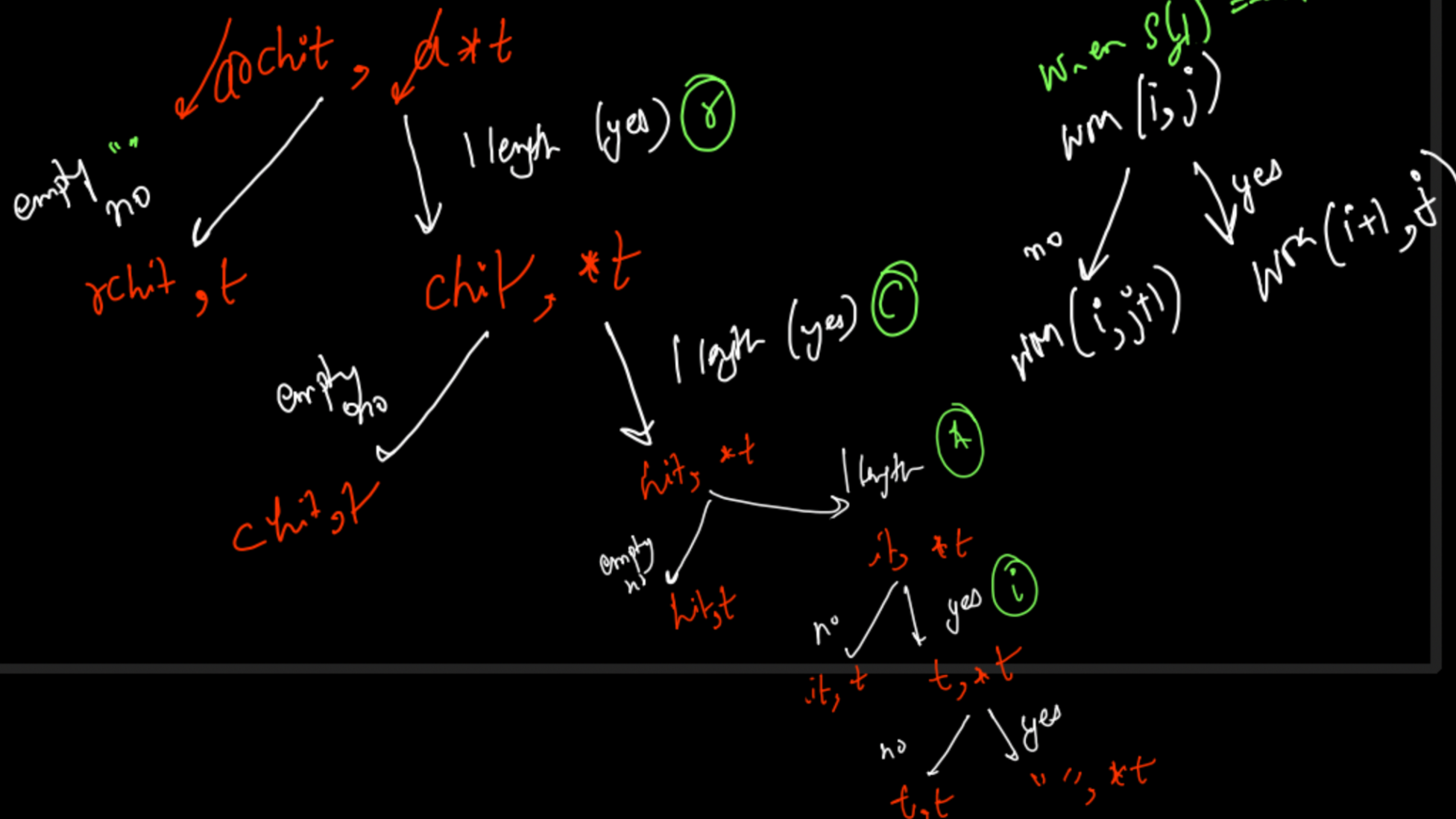
? → Single character

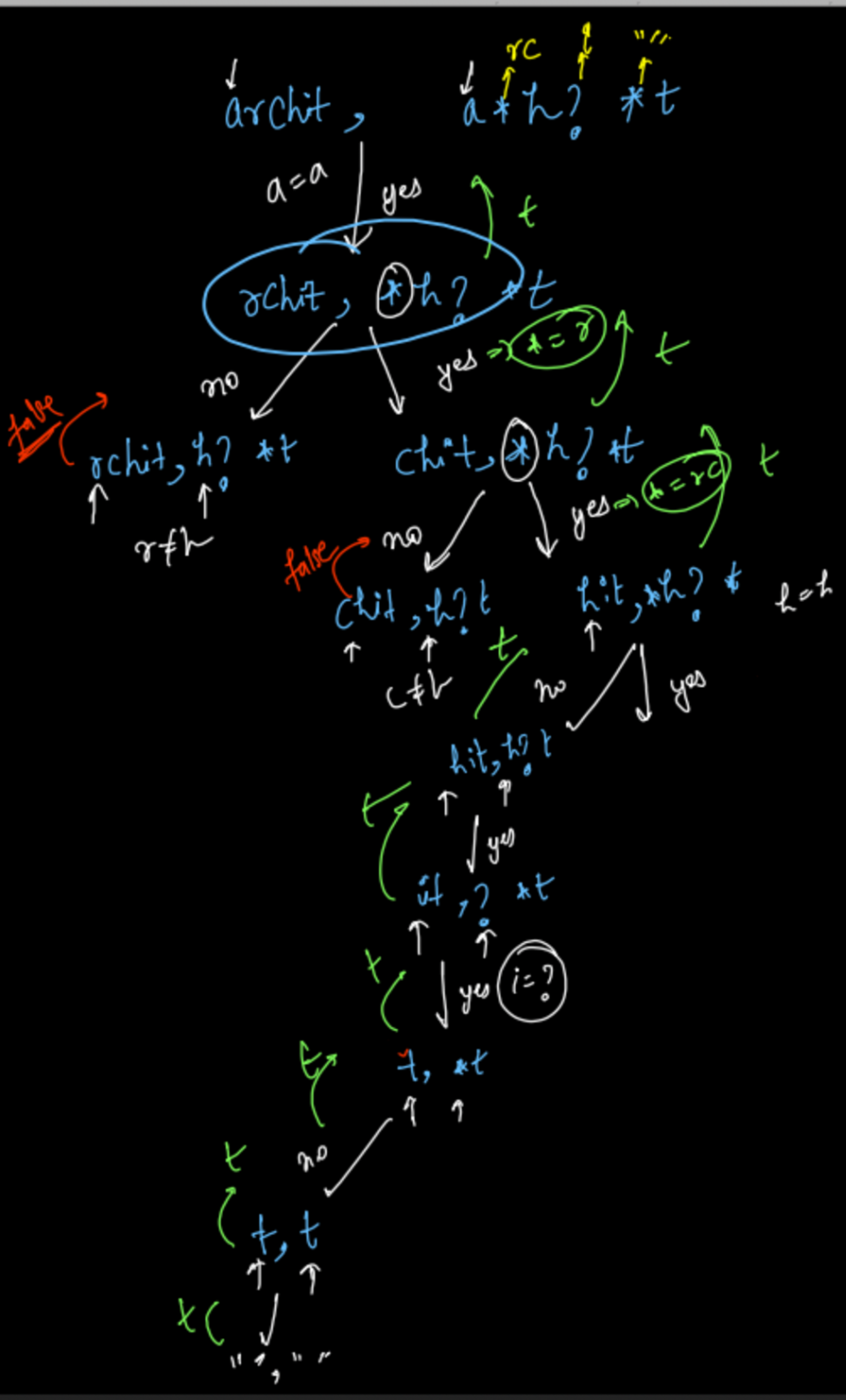
* → Contiguous (diff) group of characters

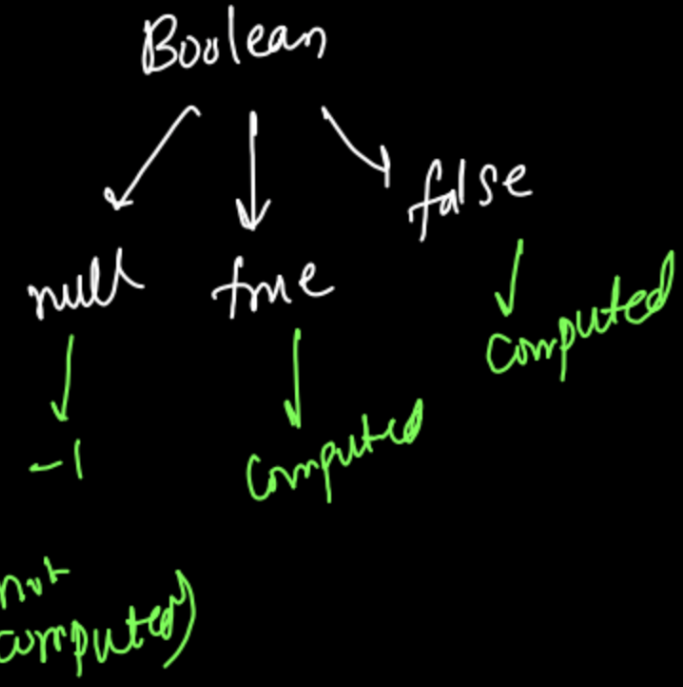
0, 1, 2, 3, ... length

s[i] == s[j]	s[j] == '?'	s[i] == '*'	s[i] != s[j]
WM(i, j) ↓ (yes) WM(i+1, j+1)	WM(i, j) ↓ ?→s[i] WM(i+1, j+1)	WM(i, j) ↙ WM(i, j+1) (no) OR ↓ yes WM(i+1, j) (yes)	return false
a = a	ar == a?		ar != ac


rchit, t
chit, t
hit, t
it, t
t, t
'', t
yes
for loop


rchit, t
chit, *t
hit, *t
it, *t
t, *t
'', *t
when s[i] == '*'
wm(i, j)
wm(i, j+1)
wm(i+1, j)


archit,
a=a
yes
archit, *h? *t
no
false
rchit, h? *t
r≠h
chit, *h? *t
yes
no
false
chit, h? t
c≠r
hit, *h? *
h=h
no
yes
hit, h? t
yes
it, ? *t
yes
i=?
t, *t
no
t, t
t(
" , "


Boolean
null
true
false
computed
-1
computed
computed
(not computed)

i=n j=m

① "" , "" → true

② "" , "a" → false

"*a" , "a*"

③ "a" , "" → false

④ "" , "**" → true

Corner Case

"" , "*"

"" , "**"

"" , "***"

```java
public boolean memo(int i, int j, String s1, String s2, Boolean[][] dp){
    if(i == s1.length() && j == s2.length()){
        return true;
    }

    if(i < s1.length() && j == s2.length()){
        // first string is not empty, second string is empty
        return false;
    }

    if(i == s1.length() && j < s2.length()){
        // first string is empty, second string is still left
        for(int k=j; k<s2.length(); k++){
            if(s2.charAt(k) != '*')
                return false;
        }
        return true;
    }
```

```java
    if(dp[i][j] != null) return dp[i][j];

    char ch1 = s1.charAt(i);
    char ch2 = s2.charAt(j);

    if(ch1 == ch2 || ch2 == '?'){
        return dp[i][j] = memo(i + 1, j + 1, s1, s2, dp);
    }

    if(ch2 == '*'){
        boolean no = memo(i, j + 1, s1, s2, dp);
        if(no == true) return dp[i][j] = true;

        boolean yes = memo(i + 1, j, s1, s2, dp);
        return dp[i][j] = yes;
    }

    // unequal characters
    return dp[i][j] = false;
}
```

```java
public boolean isMatch(String s1, String s2) {
    Boolean[][] dp = new Boolean[s1.length() + 1][s2.length() + 1];
    return memo(0, 0, s1, s2, dp);
}
```

LCS variation

↳ Time → O(N*M)

↳ Space → O(N*M)

Edit Distance

"archit" → "rohini"

(1) delete 'a'
rchit

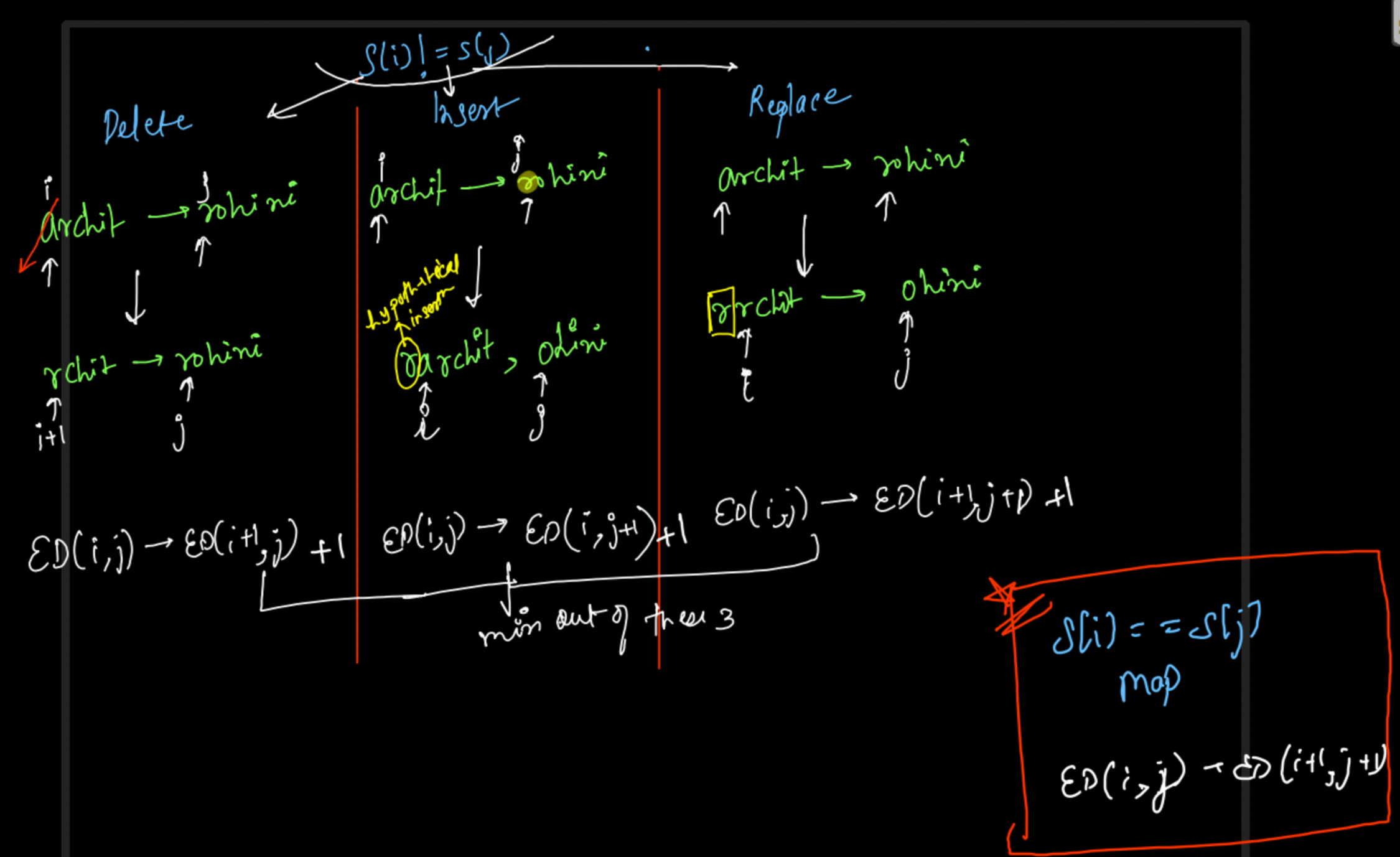
(2) replace 'c' with 'h'
rhit

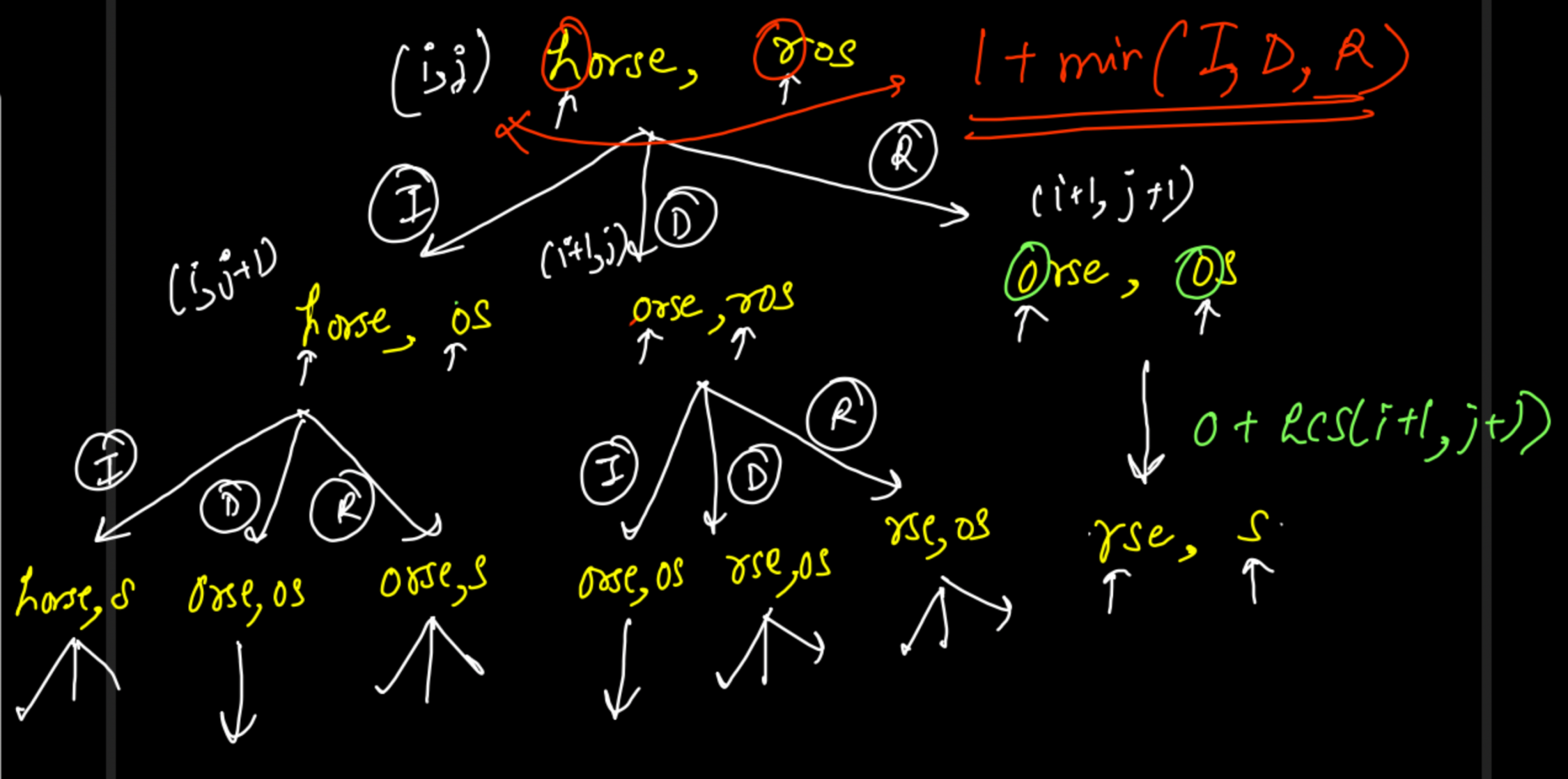
(3) replace t with n
rhin

(4) Insert i
rohini

(1) delete any char
(2) Insert any char at any char
(3) Replace any char by any char

This may ever reduce the no of operations


s(i) != s(j)
Delete
Insert
Replace
archit → rohini
rchit → rohini
i+1
j
archit → rohini
hypothetical insertion
archit → ohini
i
j
archit → rohini
rchit → ohini
i
j
ED(i,j) → ED(i+1,j) +1
ED(i,j) → ED(i,j+1)+1
ED(i,j) → ED(i+1,j+1) +1
min out of these 3
s[i] == s[j]
map
ED(i,j) → ED(i+1,j+1)


(i,j)
horse, ros
1 + min(I, D, R)
I
D
R
(i,j+1)
(i+1,j)
(i+1,j+1)
horse, os
orse, ros
orse, os
0 + LCS(i+1, j+1)
horse, s
orse, os
orse, s
orse, os
rse, os
rse, os
rse, s

Base cases

1. "", "" ⟶ 0 operations
2. "abc", "" ⟶ n operations (delete all)
3. "", "pqrs" ⟶ m operations (insert all)

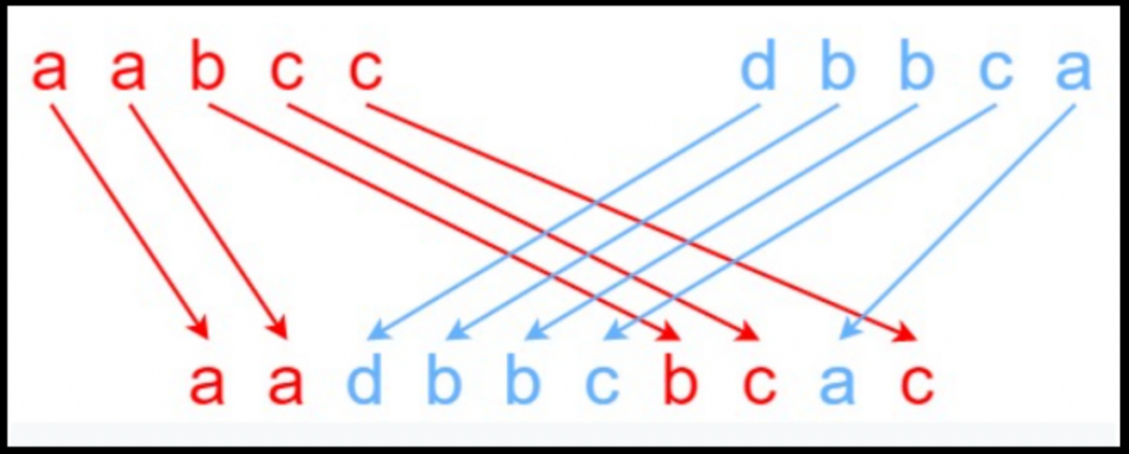
```java
public int memo(int i, int j, String s1, String s2, int[][] dp){
    if(i == s1.length() && j == s2.length()) return 0;
    if(i == s1.length()) return s2.length() - j; // First String Empty, Insert All to make Second
    if(j == s2.length()) return s1.length() - i; // Second String Empty, Delete All Char in First

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

    char ch1 = s1.charAt(i);
    char ch2 = s2.charAt(j);

    if(ch1 == ch2)
        return dp[i][j] = memo(i + 1, j + 1, s1, s2, dp);
    // no operation, since char is same

    int delete = memo(i + 1, j, s1, s2, dp);
    int insert = memo(i, j + 1, s1, s2, dp);
    int replace = memo(i + 1, j + 1, s1, s2, dp);
    return dp[i][j] =  1 + Math.min(delete, Math.min(insert, replace));
}
```

```java
public int minDistance(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);
}
```

Time → O(N * M)

Space → O(N * M)

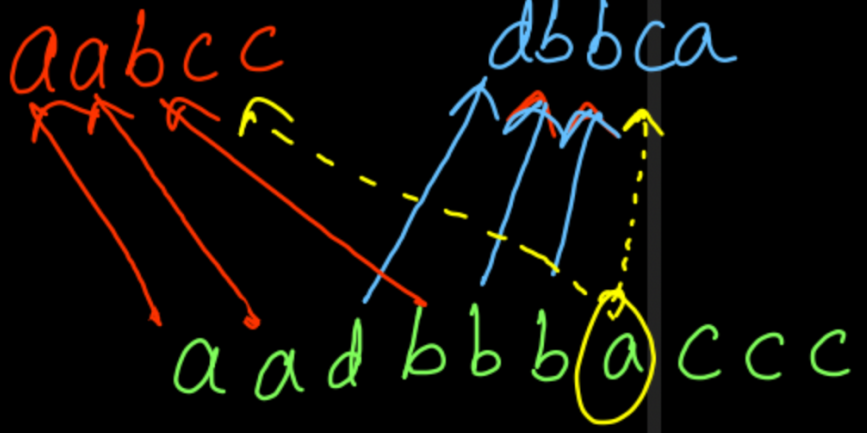
Interleaving String



S3 = n + m

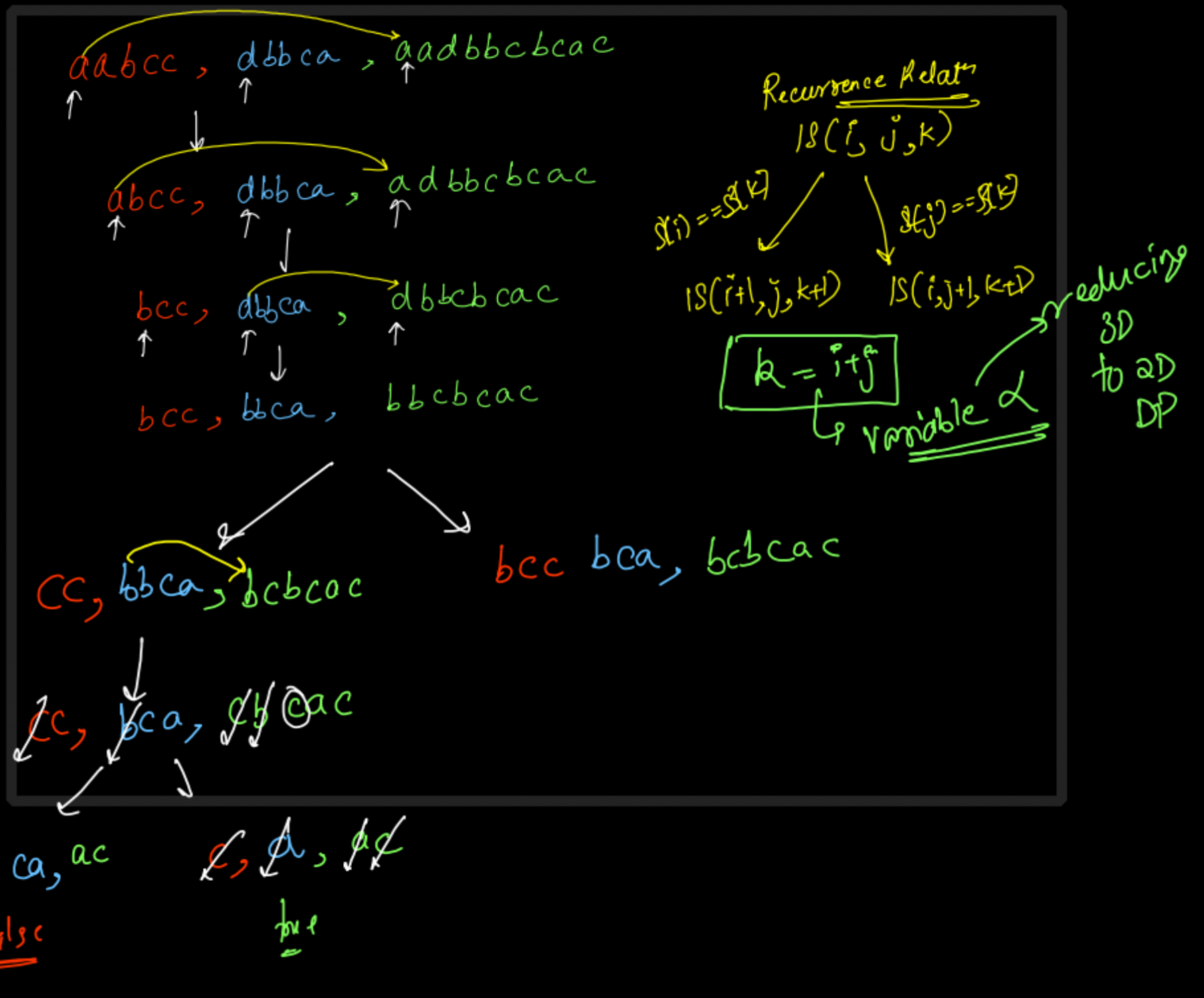
True

eg1



eg2

False


aabcc , dbbca , aadbbcbcac
abcc, dbbca, adbbcbcac
bcc, dbbca, dbbcbcac
bcc, bbca, bbcbcac
cc, bbca, bcbcac
bcc bca, bcbcac
cc, bca, cbcac
"", ca, ac
false
c, a, ac
true
Recurrence Relation
IS(i, j, k)
S(i) == S(k)
S(j) == S(k)
IS(i+1, j, k+1)
IS(i, j+1, k+1)
k = i+j
variable
reducing 3D to 2D DP

```java
public boolean memo(int i, int j, int k, String s1, String s2, String s3, Boolean[][] dp){
    // Positive Base Case
    if(k == s3.length()){
        // Result is empty
        return true;
    }

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

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

    if(s3.charAt(k) == ch1 && memo(i + 1, j, k + 1, s1, s2, s3, dp) == true){
        return dp[i][j] = true;
    }

    if(s3.charAt(k) == ch2 && memo(i, j + 1, k + 1, s1, s2, s3, dp) == true){
        return dp[i][j] = true;
    }

    return dp[i][j] = false;
}
```

```java
public boolean isInterleave(String s1, String s2, String s3) {
    if(s3.length() != s1.length() + s2.length()) return false;
    Boolean[][] dp = new Boolean[s1.length() + 1][s2.length() + 1];
    return memo(0, 0, 0, s1, s2, s3, dp);
}
```

Time ~ $O(n^2)$, Space $O(n^2)$

$k = i + j$

k is not a DP State

false ($2+1 \neq 4$)

"ar" + "c" ≠ "arch"

"ar" + "ch" ≠ "arc"

false ($2+2 \neq 3$)