Longest Common Subsequence → (L.C.S)

Given two strings. Find the length of longest common subsequence in 2 strings.

[N] s1 : a b b c d g f

[m] s2 : b a c d e g f

acdgf or bcdgf

ans = 5

[N] s1 : d e m o c r a t

[m] s2 : r e p u b l i c a n

ans = 3

B.f idea. → Consider all subsequences of s1 and s2 & then find longest common subsequence.

T.C → $O(2^N + 2^m + 2^N * N)$

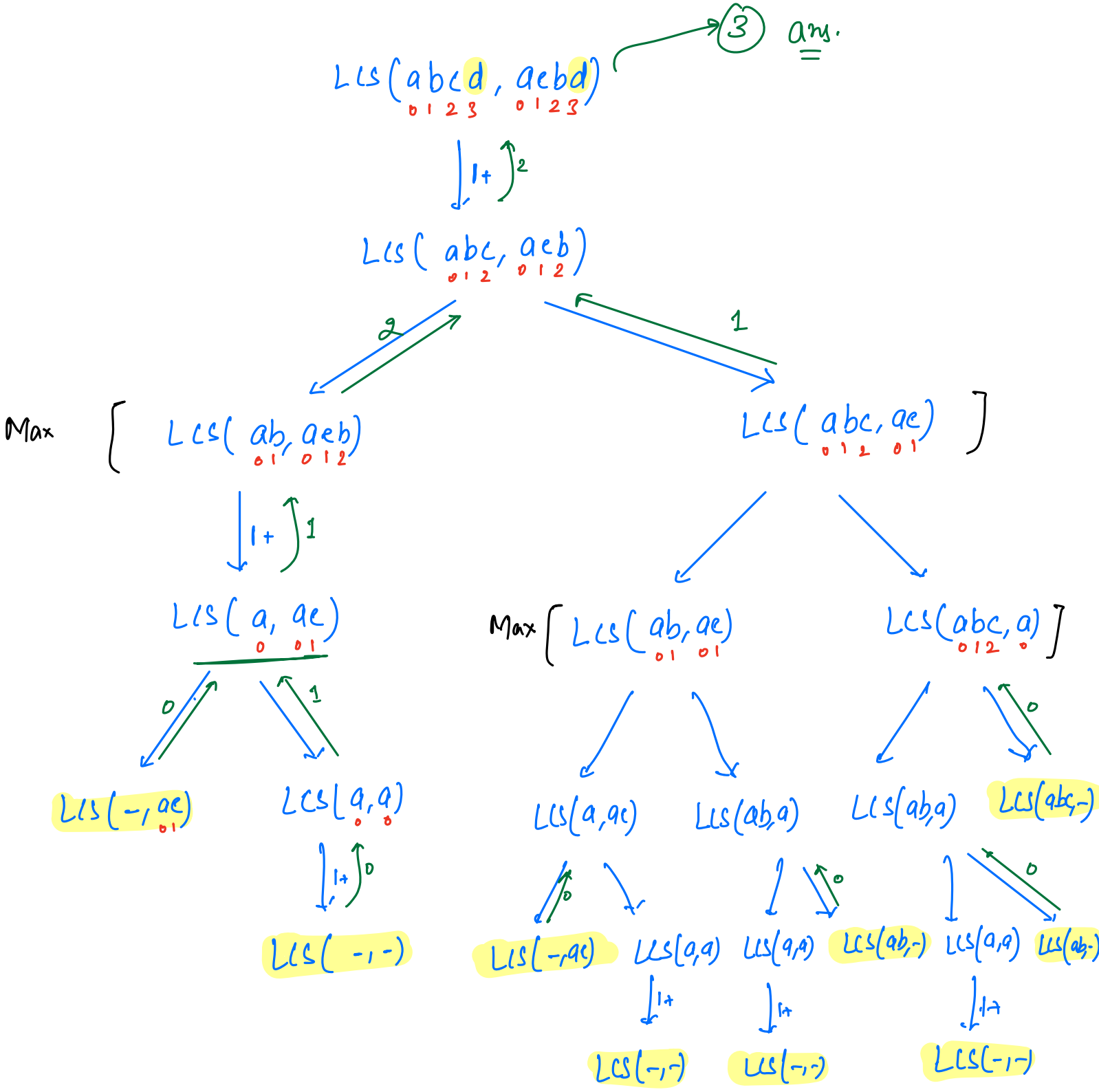
L.C.S(s1(0,N-1), s2(0,m-1))

s1[N-1] == s2[m-1]

s1[N-1] != s2[m-1]

1 + L.C.S(s1(0,N-2), s2(0,m-2))

Max { LCS(s1(0,N-2), s2(0,m-1)), LCS(s1(0,N-1), s2(0,m-2)) }



optimal substructure ✓

overlapping sub-problems ✓

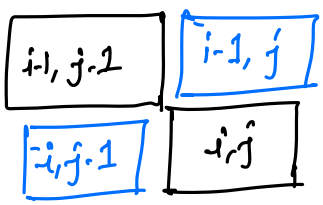
⇒ D.P ✓

code :- Top-Down Approach

```
                    n-1   m-1
                     ↓     ↓
int lcs( s1, s2, i, j , int dp[N][m] ){
    if( i < 0 || j < 0) { return 0 }
    if (dp[i][j] != -1) { return dp[i][j] }
    if( s1[i] == s2[j]){
        dp[i][j] = 1 + lcs( s1, s2, i-1, j-1, dp);
    }
    else{
        dp[i][j] = Max( lcs(s1,s2, i-1, j,dp) , lcs( s1,s2, i , j-1,dp) );
    }
    return dp[i][j];
}
```

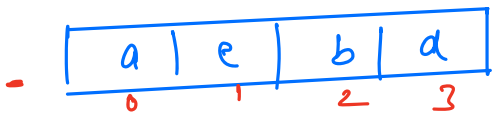
T.C → O(N*m)
S.C → O(N*m)

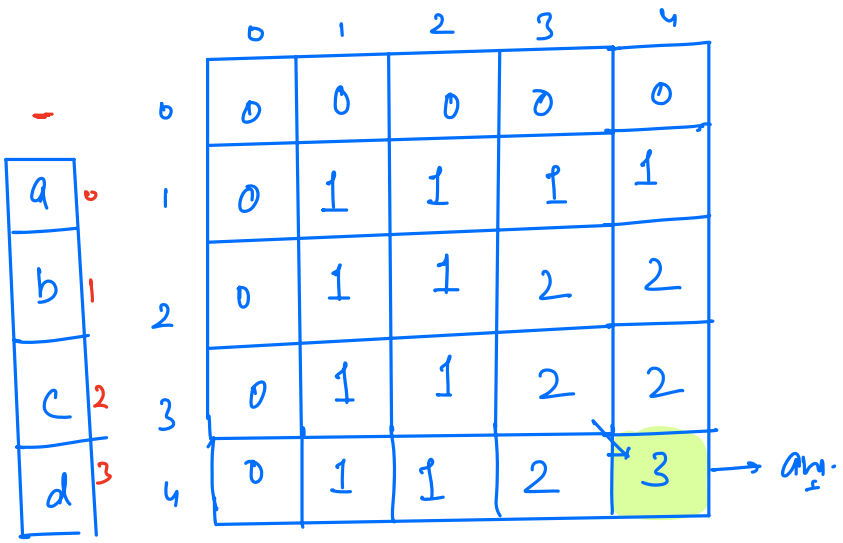
dp[i][j] ⇒ L.C.S of s1 (0,i) and s2 (0,j)



s1 → a b c d

s2 → a e b d





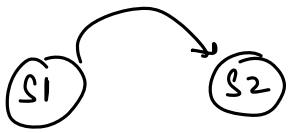
#code.→

```
dp[N+1][m+1];

initialise 0th row & 0th column with 0.

for( i=1; i<=N; i++){
    for( j=1; j<=m; j++){
        if( s1[i-1] == s2[j-1]){
            dp[i][j] = dp[i-1][j-1] + 1;
        }
        else{
            dp[i][j] = Max( dp[i-1,j], dp[i][j-1]);
        }
    }
}
return dp[N][m];
```

T.C → O(N*m)
S.C → O(N*m)

Edit- Distance

Given s1 & s2. Convert s1 → s2 by using some operations in s1 only.



① insert → C_i
② delete → C_d
③ replace → C_r

Find minimum cost to convert s1 to s2.

$C_i = 2,\ C_d = 2,\ C_r = 3$

Eg → s1 - a c (b)
s2 - a b c

ans = 2

s1 → a b c d
s2 → a b e

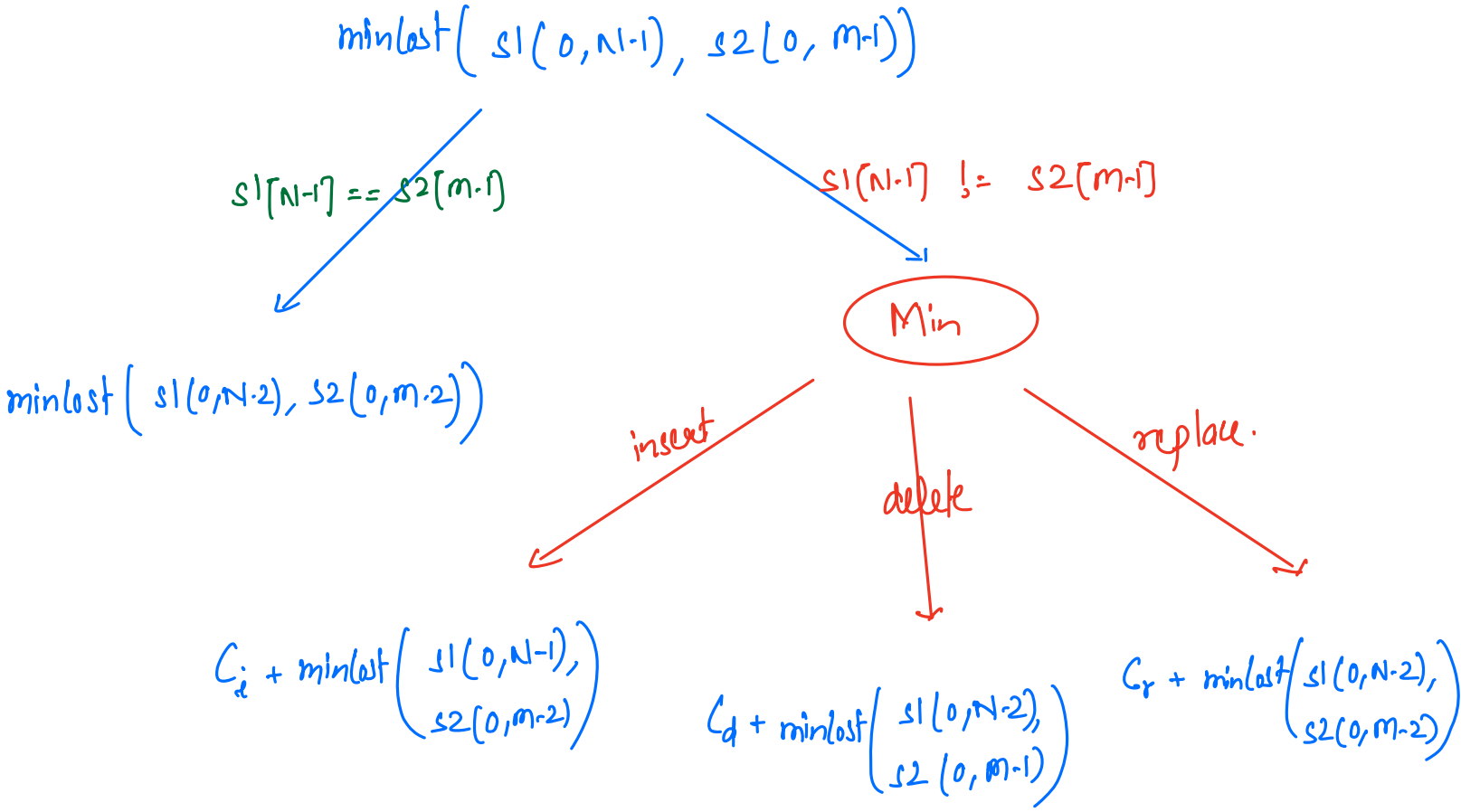
1 deletion + 1 replacement

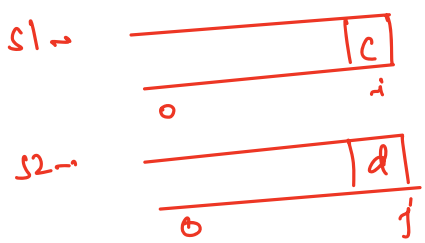
ans = 5.

s1 → a b d x y (c, g)
s2 → a b c g x

1 replacement + 1 deletion + 1 insertion

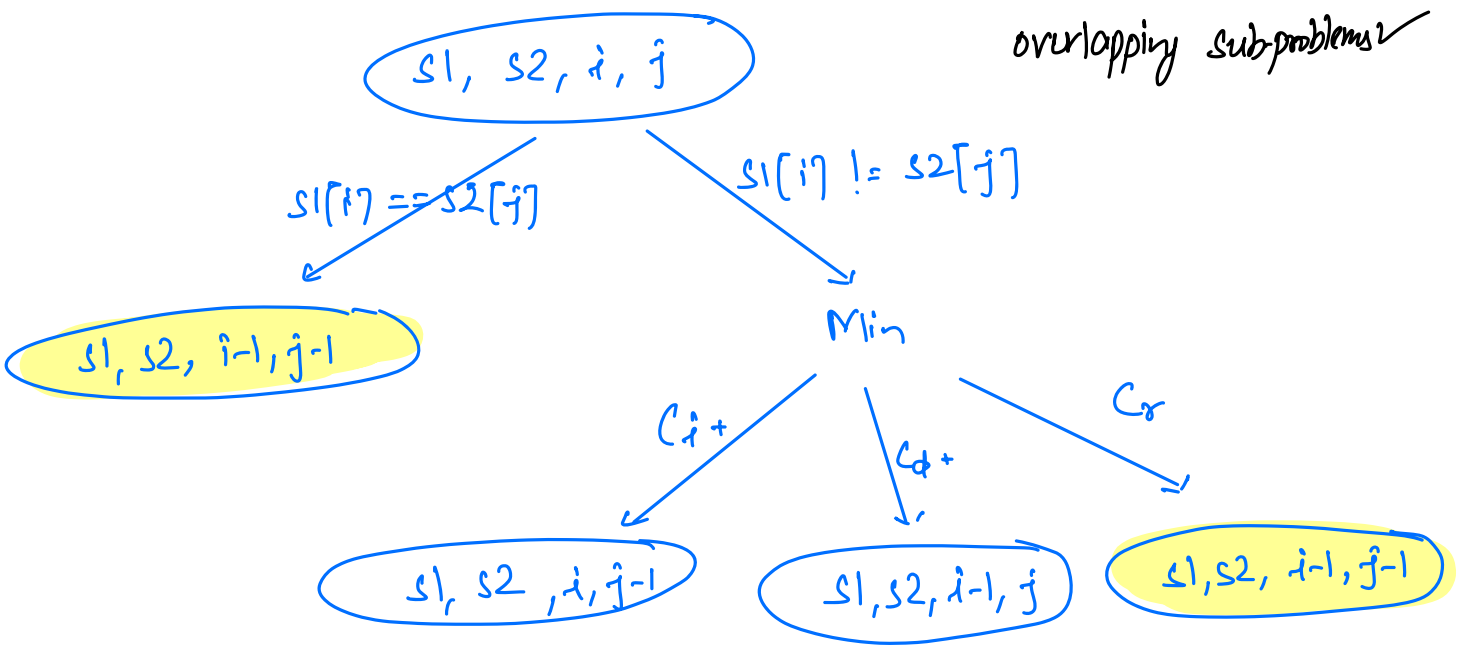
3 + 2 + 2 = (7).





optimal substructure ✓

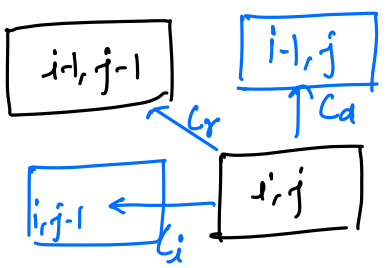
overlapping subproblems ✓



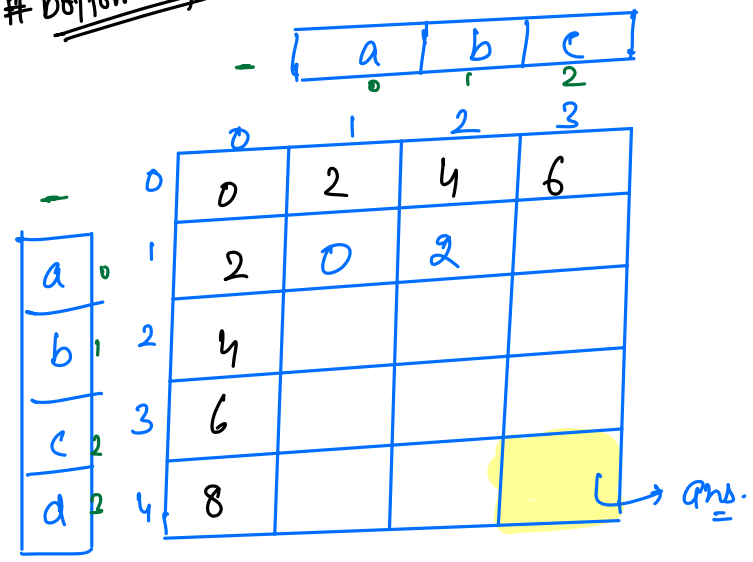
code → top-down

```
int minCost ( s1, s2, i, j , int dp[N][M] ){
    if(i < 0 && j < 0) { return 0}
    else if( i < 0) { return  Ci * (j+1) }
    else if (j < 0) { return  Cd * (i+1) }

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

    if( s1[i] == s2[j]){
        dp[i][j] = minCost( s1, s2, i-1, j-1 , dp);
    }
    else{
        dp[i][j] = Min { Ci + minCost( s1, s2, i, j-1 ,dp);
                         Cd + minCost(s1, s2, i-1, j ,dp);
                         Cr + minCost(s1, s2, i-1, j-1, dp); }
    }
    return dp[i][j];
}
```

T.C → O(N*m)
S.C → O(N*m)



bottom-up.



Ci → 2

Cd → 2

Cr → 3

#code - todo.

Wildcard Pattern matching

Given s1 & s2. Check if they are matching.

s2 → it can contain '?', '*'

'?' → matches with any single character.

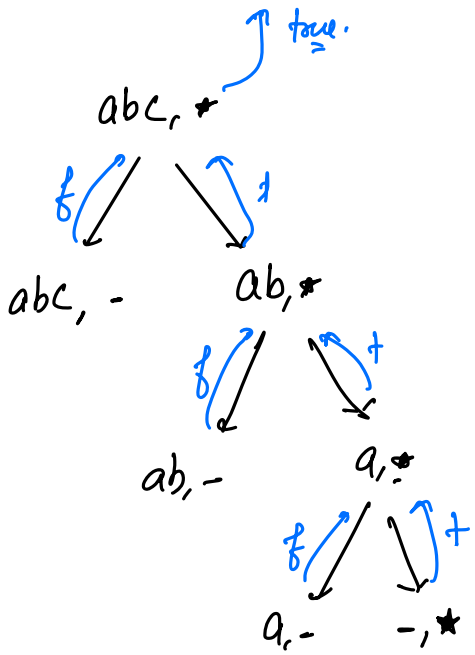
'*' → matches with 0 or more characters.

① s1 → a b a c d — true.

s2 → a b a c d

② s1 → a b a c d — true

s2 → a ? a c ?



③ s1 → x b b z z c — true

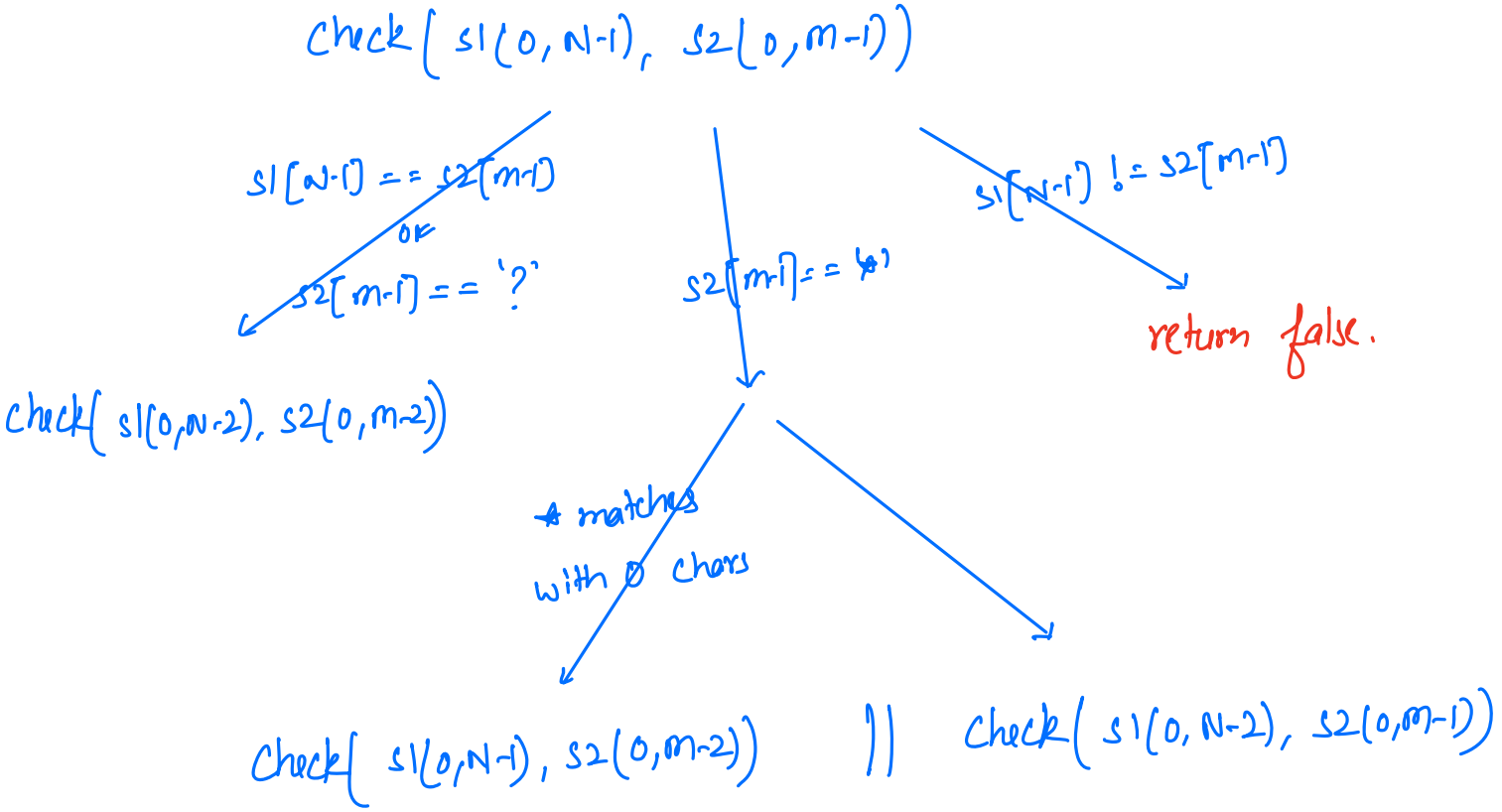
s2 → x * z *

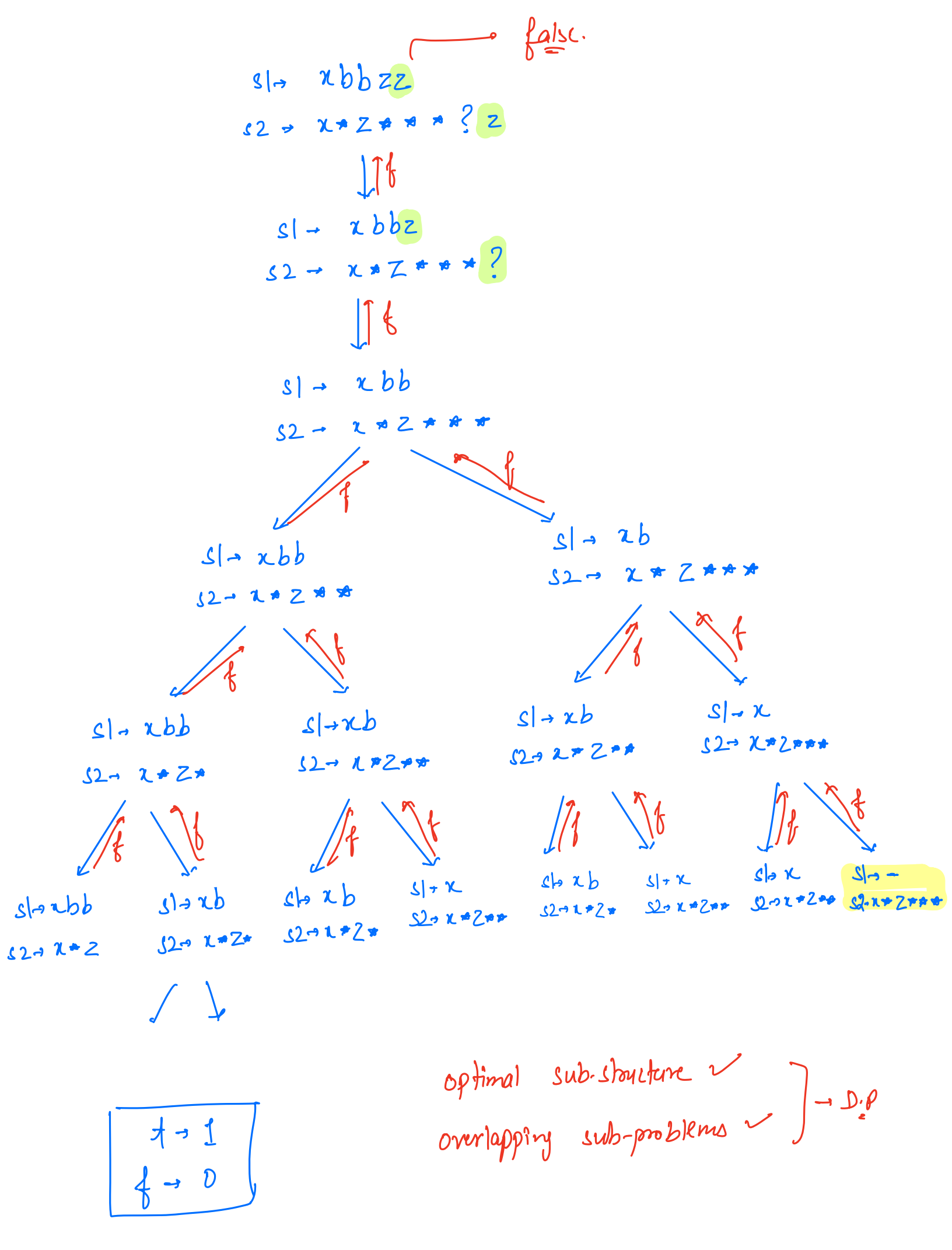
④ s1 → x b b z z — false.

s2 → x * z * * * ? z

⑤ s1 → x b b z z — true.

s2 → x * z * * * ?


check(s1(0, N-1), s2(0, m-1))
s1[N-1] == s2[m-1]
or
s2[m-1] == '?'
s2[m-1] == '*'
s1[N-1] != s2[m-1]
return false.
check(s1(0, N-2), s2(0, m-2))
* matches with 0 chars
check(s1(0, N-1), s2(0, m-2)) || check(s1(0, N-2), s2(0, m-1))


false.
s1→ xbbzz
s2→ x*z***?z
f
s1→ xbbz
s2→ x*z***?
f
s1→ xbb
s2→ x*z***
f
f
s1→ xbb
s2→ x*z**
s1→ xb
s2→ x*z***
f
f
f
s1→ xbb
s2→ x*z*
s1→xb
s2→ x*z**
s1→ xb
s2→ x*z**
s1→ x
s2→ x*z***
f
f
f
f
f
f
s1→ xbb
s2→ x*z
s1→ xb
s2→ x*z*
s1→ xb
s2→ x*z*
s1→ x
s2→ x*z**
s1→ xb
s2→ x*z*
s1→ x
s2→ x*z**
s1→ x
s2→ x*z**
s1→ -
s2→ x*z***
t → 1
f → 0
optimal sub-structure ✓
overlapping sub-problems ✓
→ D.P

top-down → code.

```
int check ( s1, s2, i, j , dp[N][m] ){

    if( i < 0 && j < 0) { return 1 }
    else if( i < 0 && checkstars (s2, j)) {
    {     return 1;
    }
    else if ( i < 0 || j < 0 ) { return 0 }

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

    if ( s1[i] == s2[j] || s2[j] == '?' ){
    {     dp[i][j]= check ( s1, s2, i-1, j-1, dp);
    }
    else if ( s2[j] == '*'){
    |
    |     dp[i][j]= max { check( s1, s2, i, j-1, dp)
    |                    check( s1, s2, i-1, j, dp) }
    |
    }
    else{
    {     dp[i][j]= 0;
    }

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

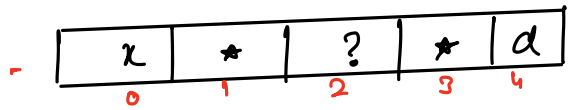
T.C → O(N*m)
S.C → O(N*m)

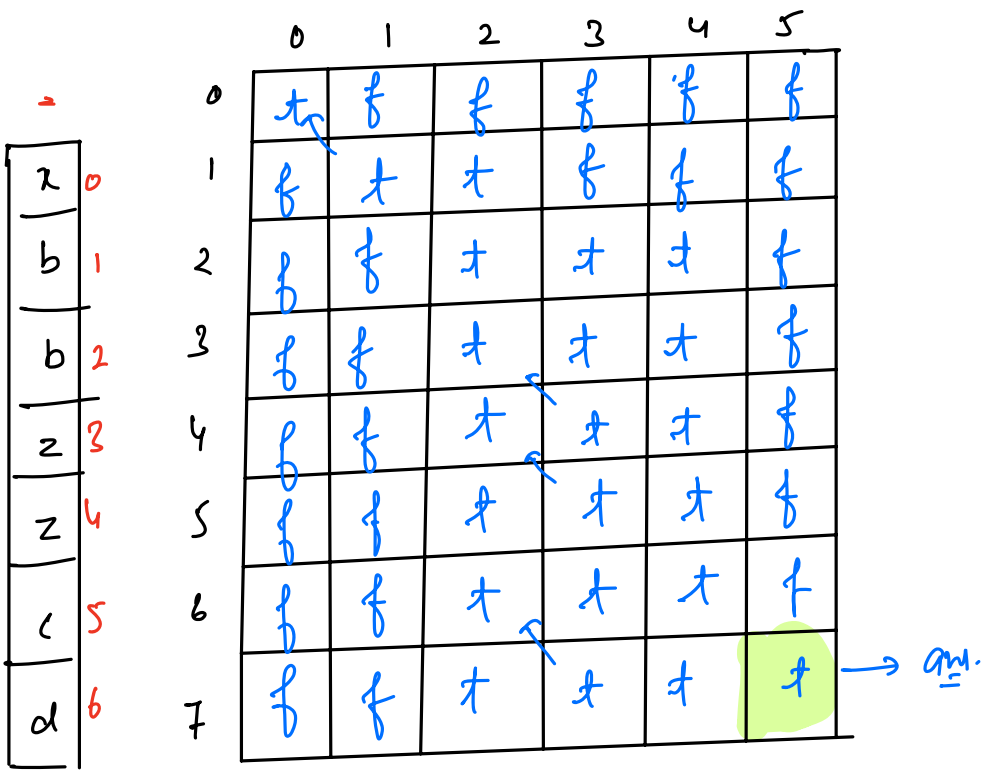
```
boolean checkStars( s2, j){
    for( i=0; i<=j; i++){
        if( s2[j] != '*'){ return false }
    }
    return true;
}
```

#bottom-up

s1 → x b b z z c d

s2 → x * ? * d





```
dp[N+1][m+1]

dp[0][0] = true;

for( j = 1 ; j ≤ m ; j++){
    if( s2[j-1] == '*' ){
        dp[0][j] = dp[0][j-1];
    }
    else{
        dp[0][j] = false;
    }
}

for( i = 1; i ≤ N; i++){
    dp[i][0] = false;
}

for( i = 1 ;  i ≤ N; i++){
    for( j = 1; j ≤ m; j++){
        if( s1[i-1] == s2[j-1] || s2[j-1] == '?' ){
            dp[i][j] = dp[i-1][j-1];
        }
        else if( s2[j-1] == '*' ){
            dp[i][j] = dp[i-1][j] || dp[i][j-1];
        }
        else{
            dp[i][j] = false;
        }
    }
}
return dp[N][m];
```

T.C → O(N*m)
S.C → O(N*m)

⇒ Regular Expression Matching.

arr(i)

(2, 3, 5, 7) → Sum. 100