11/9/2023

DP-5

① Given 2 strings. Find length of longest common subsequence in the 2 strings.

S1 → a b b c d g f
S2 → b a c d e g f

acdgf
bcdgf
} ans = 5

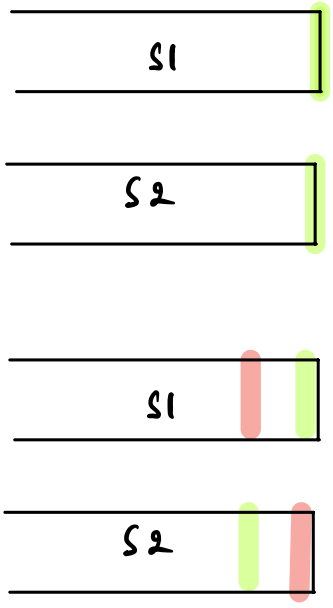
S1 → k l a g u i p
S2 → l g i g k m

lgi
} ans = 3

Bruteforce: Generate all subsequence of one string & compare with the other string.

a b c d e
a c d b

S1 → a b c d e
S2 → b c d e a
} ans = bcde



LCS(S1(0-(n-1)), S2(0-(m-1))

S1 → 0-(n-1)
S2 → 0-(m-1)

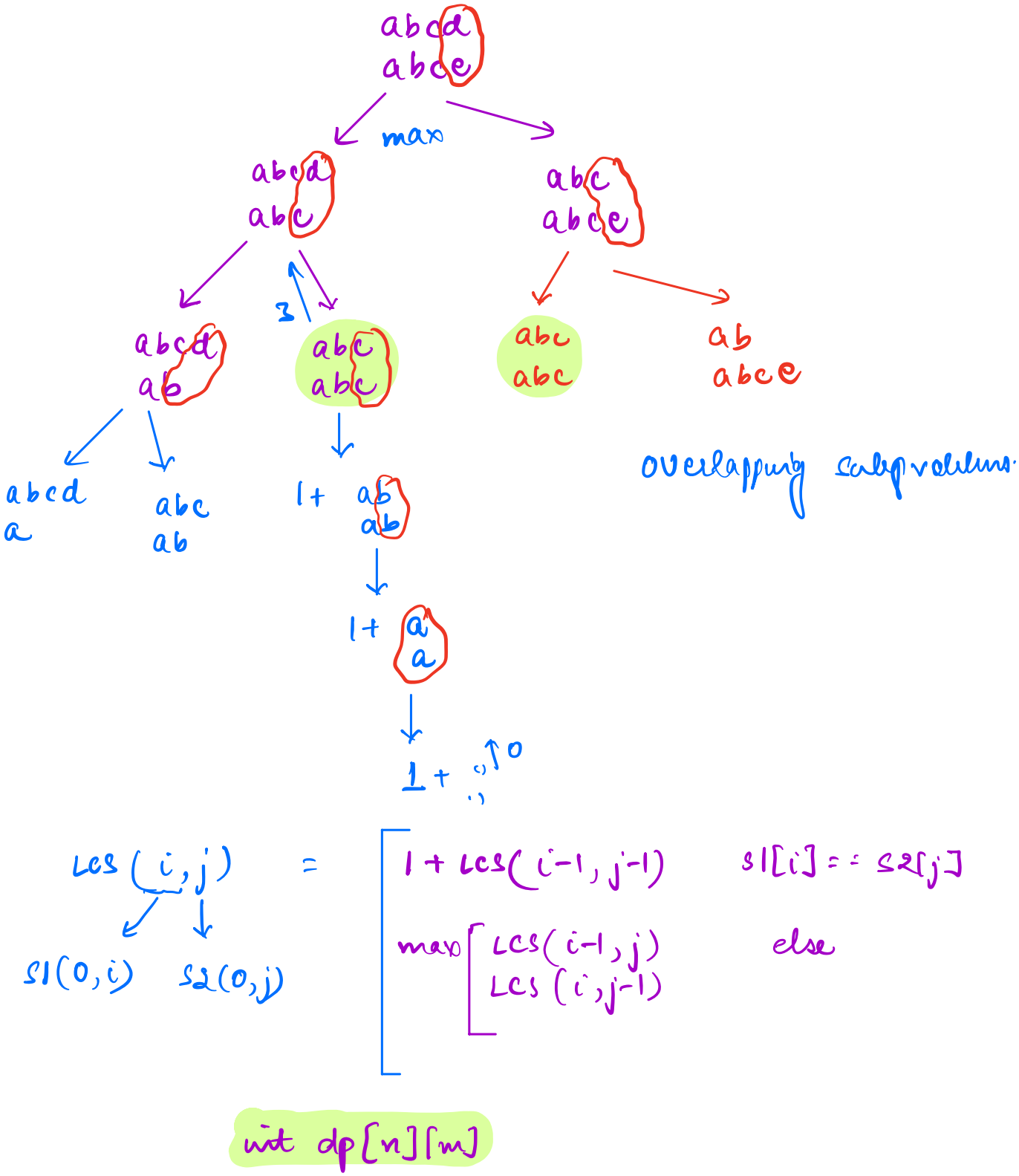
S1[n-1] == S2[m-1]

LCS(S1(0-(n-2)), S2(0-(m-2)) + 1

not equal

max {
LCS(S1(0-(n-1)), S2(0-(m-2))
LCS(S1(0-(n-2)), S2(0-(m-1))
}

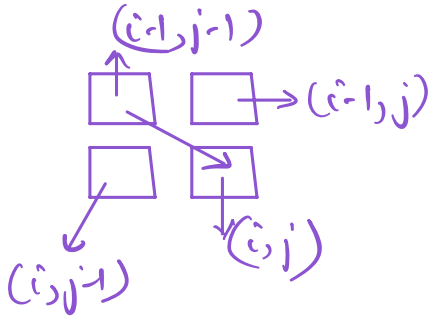
optimal substructure.


abcd
abce
max
abcd
abc
abc
abce
3
abcd
ab
abc
abc
abc
abc
ab
abce
overlapping subproblems
abcd
a
abc
ab
1+ ab
ab
1+ a
a
1+ 0
LCS(i,j) =
S1(0,i) S2(0,j)
1 + LCS(i-1, j-1) S1[i] == S2[j]
max LCS(i-1, j)
LCS(i, j-1)
else
int dp[n][m]

Code

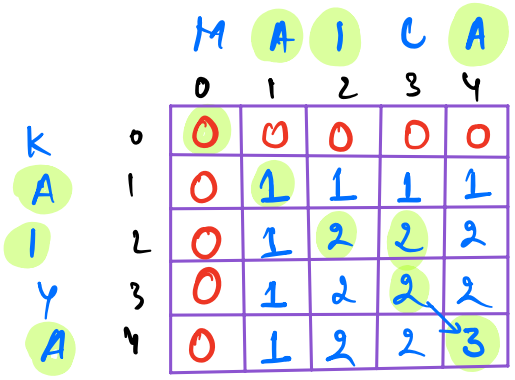
```
int dp[N][M] ; initialize -> -1

int lcs (s1, s2, i, j) {
    if(i<0 || j<0) return 0; { if either of the strings become empty }
    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);
    }

    else {
        dp[i][j] = max( lcs(s1, s2, i-1, j),
                        lcs(s1, s2, i, j-1));
    }
    return dp[i][j];
}
```

TC: O(N*M)
SC: O(N*M)



S1 → K A I Y A

S2 → M A I C A



dIA → ans.

Edit distance

Make $s1 \rightarrow s2$

from anywhere
- insert → $cost_i$
- delete → $cost_d$
- replace → $cost_r$

Find min cost to convert s1 to s2.

s2 cannot be changed, only s1 can be modified.

$cost_i = 2$
$cost_d = 2$
$cost_r = 3$

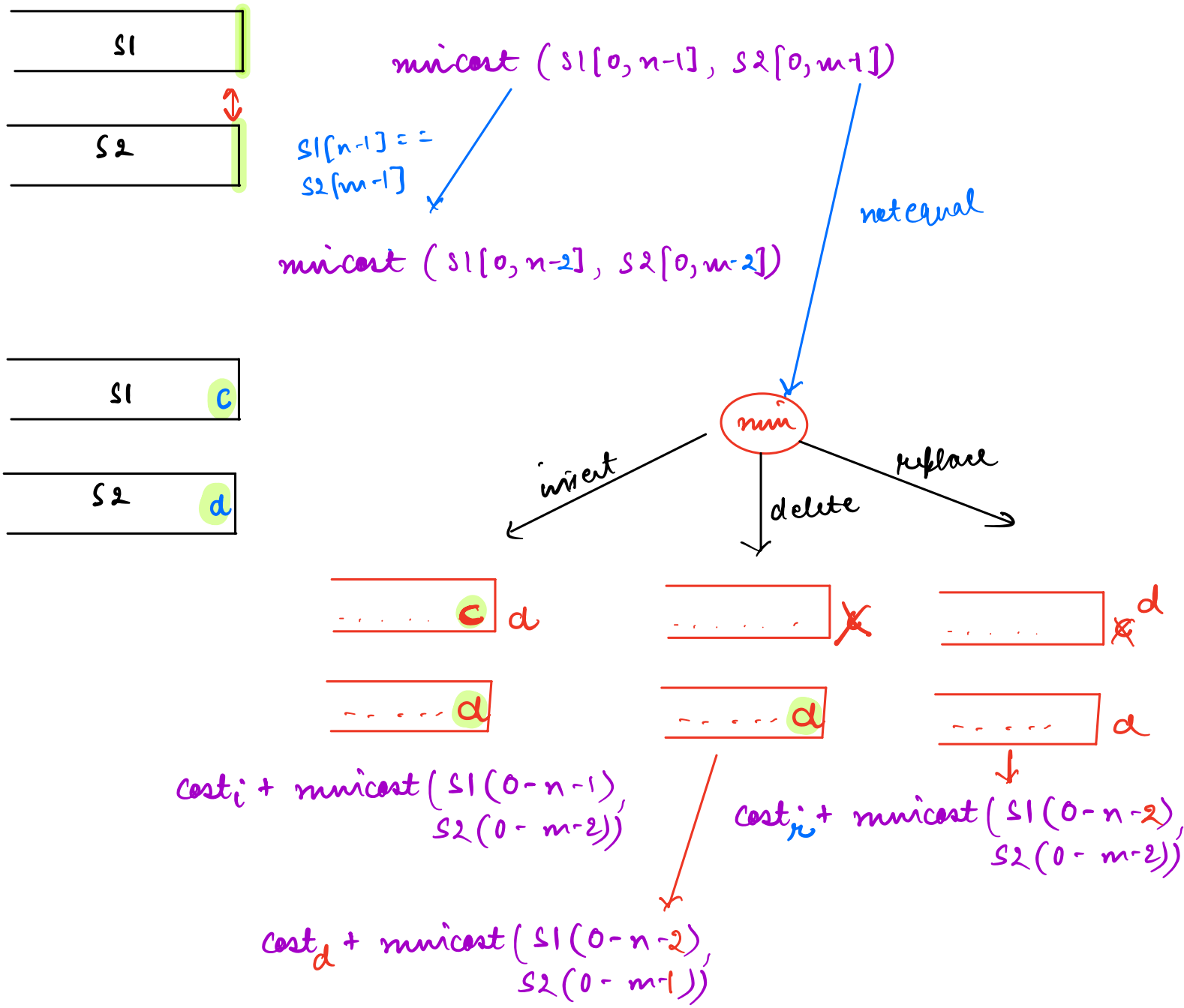
① s1 → a b c
s2 → a b c
ans = 2

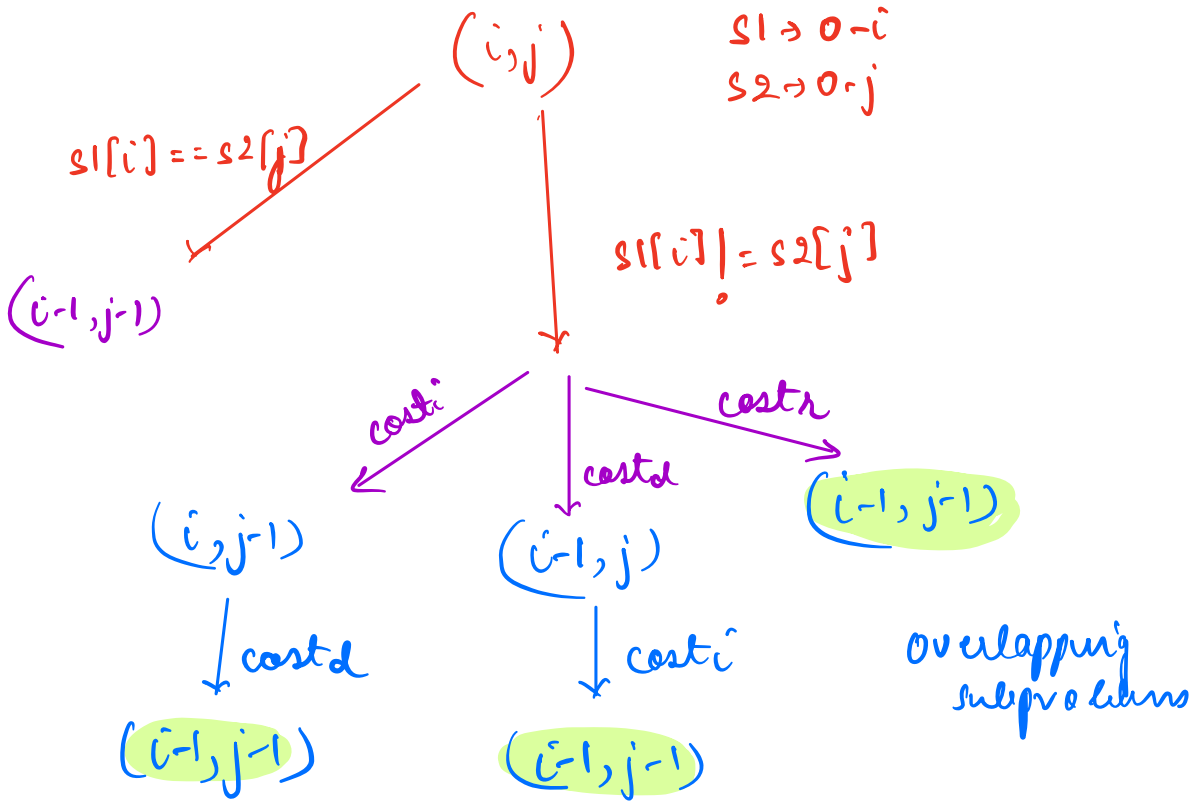
② s1 → a b c d
s2 → a b e
opt 1 → D + D + I → 2+2+2 = 6

opt 2 → R + R → 3+3 = 6

③ s1 → a b c d x y
s2 → a b c g x

I + R + D = 2 + 3 + 2 ⇒ 7


S1
S2
micost (S1[0, n-1], S2[0, m-1])
S1[n-1] == S2[m-1]
micost (S1[0, n-2], S2[0, m-2])
notequal
S1 c
S2 d
min
insert
delete
replace
c d
d
d
d
d
Cost_i + micost (S1(0-n-1), S2(0-m-2))
Cost_d + micost (S1(0-n-2), S2(0-m-1))
Cost_i + micost (S1(0-n-2), S2(0-m-2))


(i,j)
s1 → 0-i
s2 → 0-j
s1[i] == s2[j]
(i-1, j-1)
s1[i] != s2[j]
costi
costr
costd
(i, j-1)
(i-1, j)
(i-1, j-1)
costd
costi
(i-1, j-1)
(i-1, j-1)
overlapping subproblems

$(0-j) \rightarrow R-L+1$
$\rightarrow j-0+1$
$\rightarrow j+1$

```
if (i<0 && j<0) return 0;
if (i<0) { // only option is to insert remaining
    return costi * (j+1);
}
else if (j<0) {
    return costd * (i+1);
}
if (dp[i][j] != -1) return dp[i][j];
```

TC: O(N*M)
SC: O(N*M)

$$dp[i][j] = \begin{cases} dp[i-1][j-1] & \text{if } s1[i] = s2[j] \\ \min \begin{cases} cost_i + dp[i][j-1] \\ cost_d + dp[i-1][j] \\ cost_r + dp[i-1][j-1] \end{cases} & \text{else} \end{cases}$$

Meet at 8:50 am IST

Wildcard pattern matching

check if s1 matches s2 pattern.

(1) s: a b a c d
p: a b a c d ✓

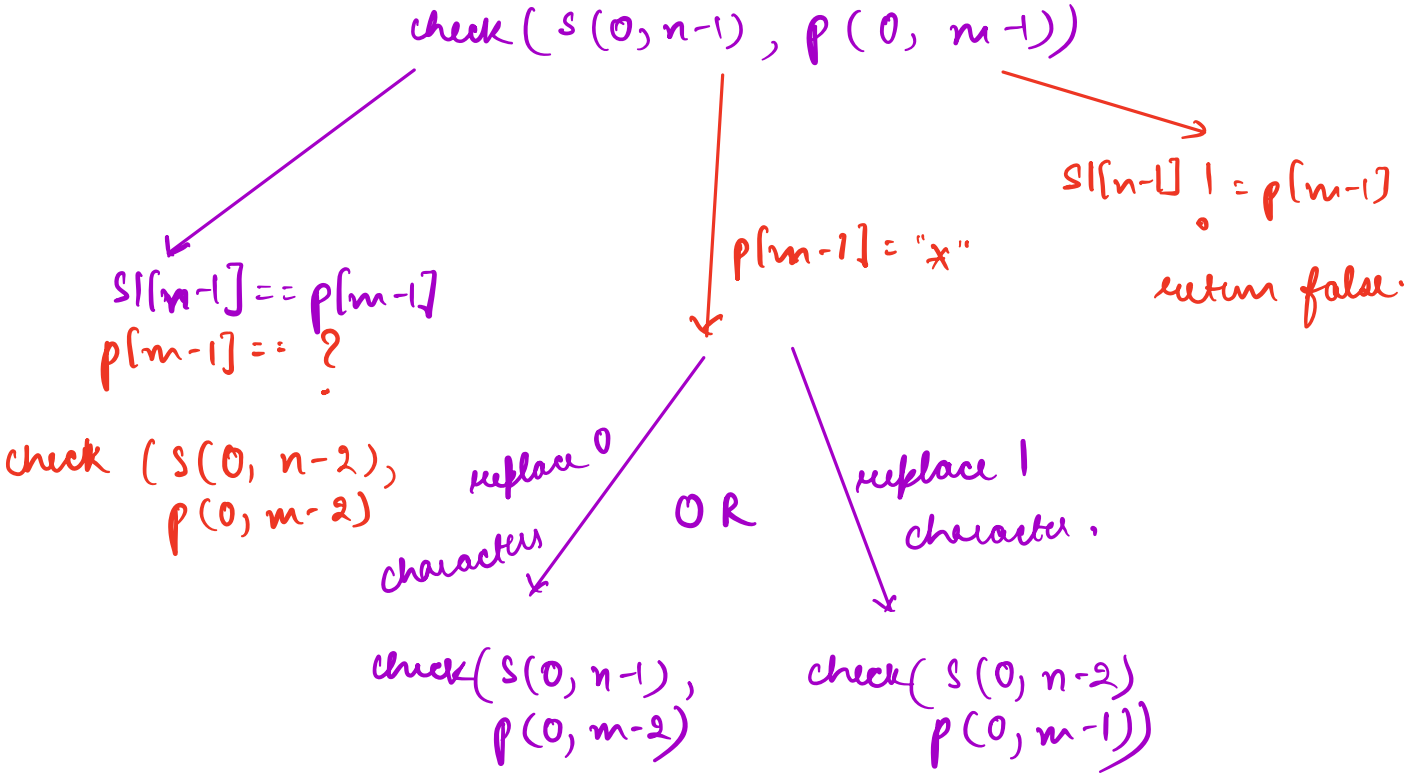
(2) s: a b a c d
p: a ? a ? d ✓
? → can be replaced by exactly 1 character.

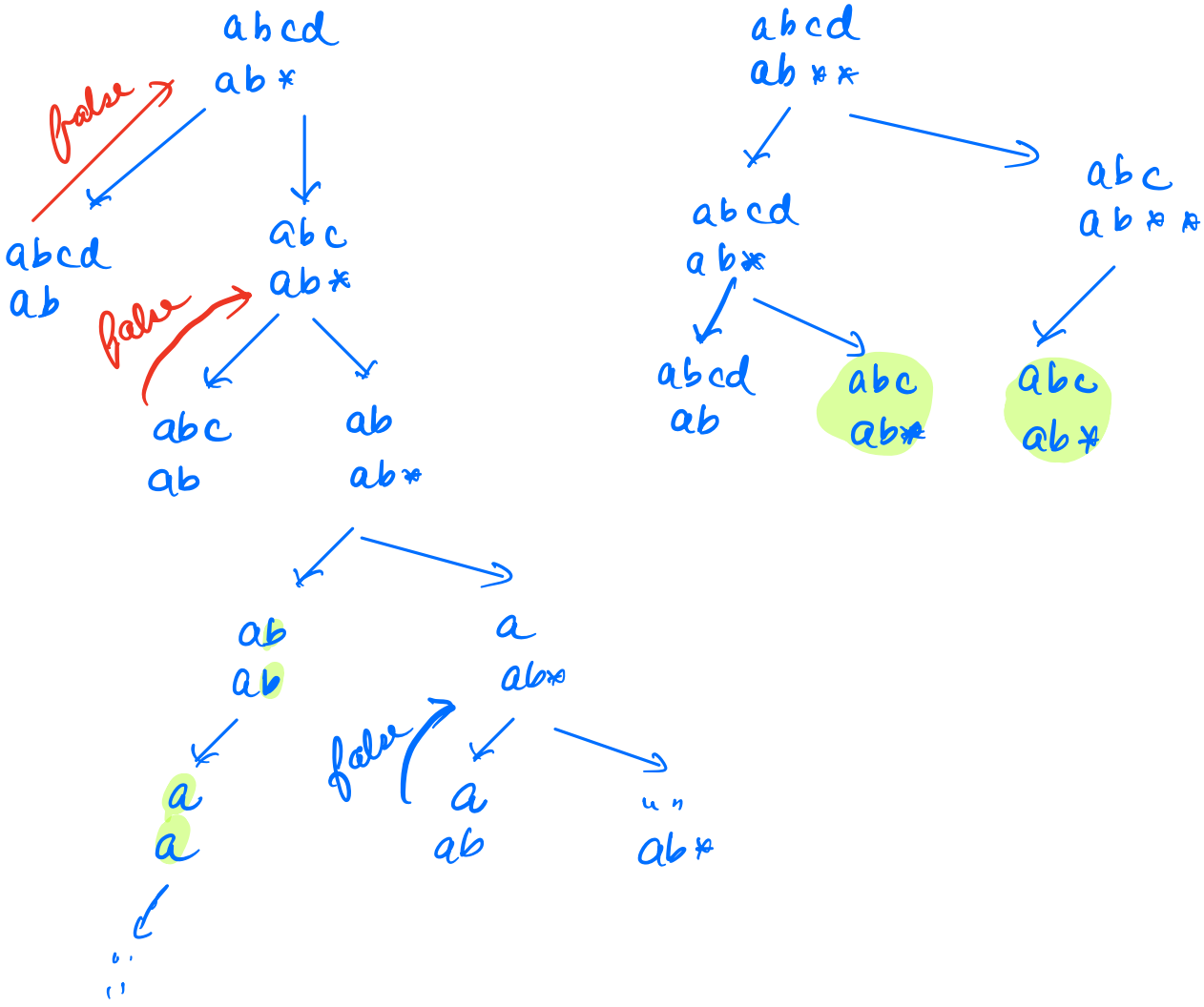
(3) s: a b b a c
p: a * c ✓
* → can be replaced by 0 or more characters (seq)

(4) s: x b b z z c
p: x * z * x ✗

(5) s: x b b z z c
p: x * z * * ✓

(6) s: x b b z z
p: x * z * * * ? z ✗


check (s(0, n-1) , p(0, m-1))
s[n-1] != p[m-1]
return false.
s[n-1] == p[m-1]
p[m-1] == ?
p[m-1] = "*"
check (s(0, n-2), p(0, m-2)
replace 0 character
OR
replace 1 character.
check(s(0, n-1), p(0, m-2)
check(s(0, n-2) p(0, m-1))


abcd
ab*
false
abcd
ab
abc
ab*
false
abc
ab
ab
ab*
ab
ab
a
ab*
a
a
false
a
ab
" "
ab*
abcd
ab**
abcd
ab*
abcd
ab
abc
ab*
abc
ab**
abc
ab*

$$dp[i][j] = \begin{cases} dp[i-1][j-1] & \text{if } s1[i] == p[j] \text{ or } p[j] == ? \\ dp[i-1][j] \;||\; dp[i][j-1] & \text{else if } p[j] == '*' \\ \text{return false} & \text{else if } s[i] != p[j] \end{cases}$$

```
if (i<0 && j<0) return true;
else if (j<0) return false
else if (i<0) {
    {
        if only * remaining → true
        else
            false
    }
```

s1 = " "
p = ***

Doubts

key → value , i-j → boolean.

1-2 →