DP on Strings

~~~~~~~~

Agenda

~~~~~~~~

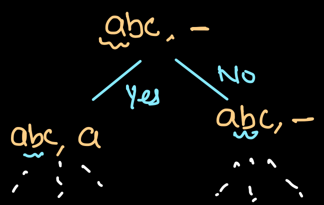
1. Longest Common Subsequence (LCS)
2. Edit Distance
3. Wildcard Pattern Matching

Hello Everyone
Very Special Good Evening
to all of you 😊😊😊
We will start session
from 9:06 PM

abc → count = 8

–, a, b, c, ab, ac, bc, abc

length → comt → 2^n



~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Longest Common Subsequence (LCS)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Given two strings S1 and S2. Find the length of common Subsequence in these strings.
^ longest

[N] S1: a b b c d g f

[M] S2: b a c d e g f

→ common ⇒ bcdgf
OR
acdgf

→ length = 5 Ans

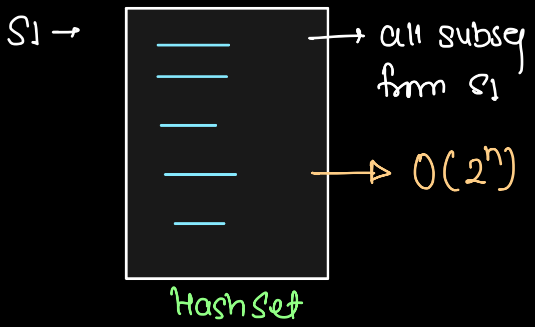
[N] S1: d e m o c r a t

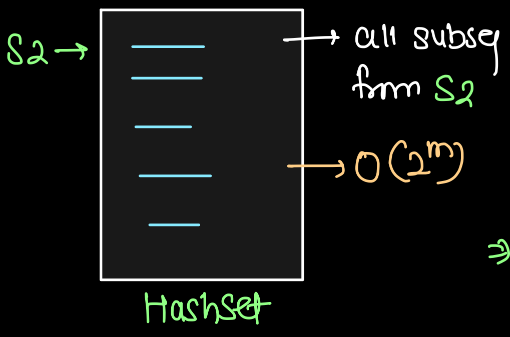
[M] S2: r e p u b l i c a n

→ longest common ⇒ eca

length = 3 Ans

Bruteforce Idea: Consider all subsequences of S1 and S2 and then find the longest common subseq.





$O(2^n + 2^m + 2^n * \underbrace{l}_{\text{Hash code}})$

$\Rightarrow O(2^n + 2^m + n * 2^n)$

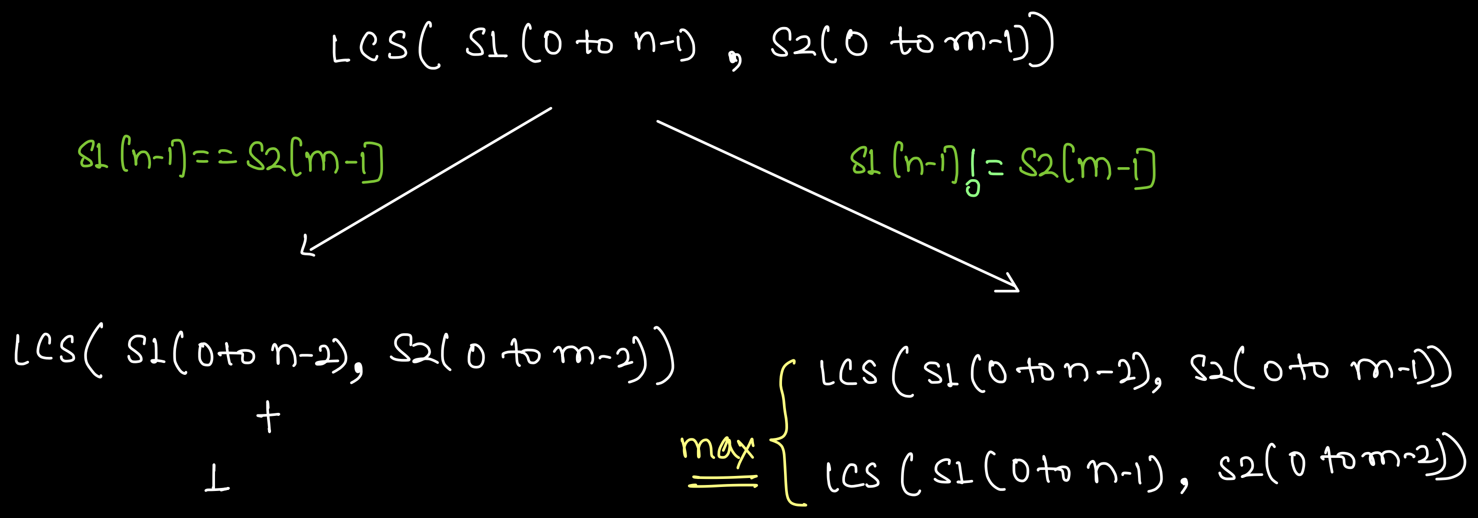
Note: 9f we will select last index, it will give feel of top-down approach, otherwise we can start from first also

[N] S1: a b b c d g f

[M] S2: b a c d e g f

[N] S1: d e m o c r a t

[M] S2: r e p u b l i c a n


LCS(S1(0 to n-1) , S2(0 to m-1))
S1(n-1)==S2(m-1)
S1(n-1)!=S2(m-1)
LCS(S1(0 to n-2), S2(0 to m-2))
+
1
max
LCS(S1(0 to n-2), S2(0 to m-1))
LCS(S1(0 to n-1), S2(0 to m-2))

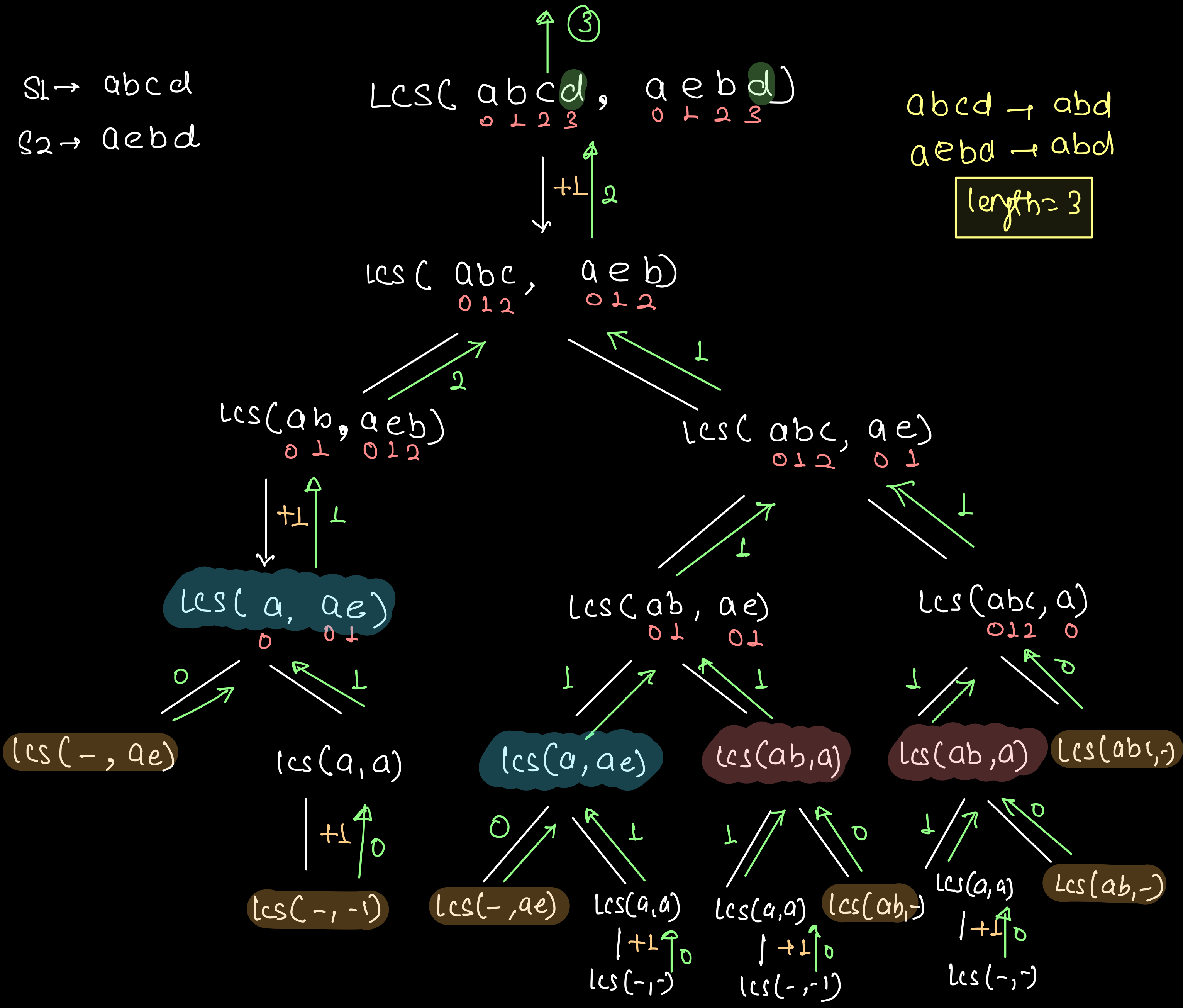
S1 → abcd

S2 → aebd

abcd → abd

aebd → abd

length= 3



# code.

```
dp[N][M] , ∀ i,j  dp[i][j] = -1;
```

└→ globally created so, it is accessible.

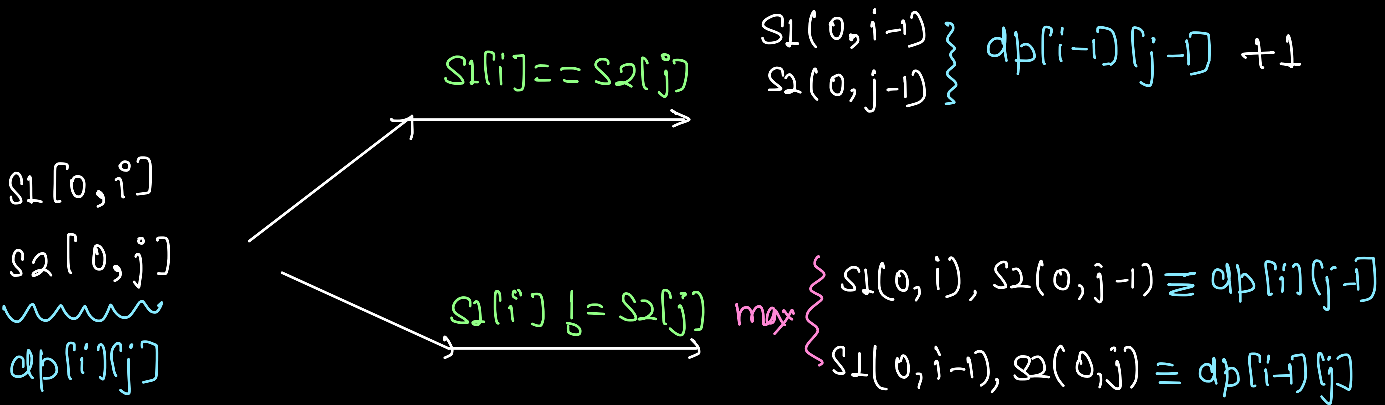
Initial → n-1 m-1

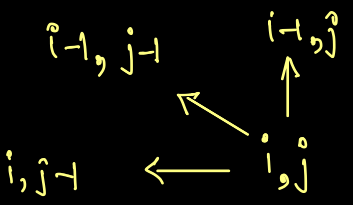
```
int lcs ( String s1, String s2, i, j ){
    if( i<0 || j<0){ return 0;}
    if( dp[i][j] != -1){ return dp[i][j];}

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

T.C: O(n*m)

S.C: O(n*m)





S1 → a b c d   dp[N+1][m+1]

S2 → a e b d

dp[i][j] → String s1 from 0 to i-1
String s2 from 0 to j-1

			–	a	e	b	d
				0	1	2	3
			0	1	2	3	4
–		0	0	0	0	0	0
a	0	1	0	1	1	* 1	1
b	1	2	0	1	1	2	2
c	2	3	0	1	1	2	2
d	3	4	0	1	1	2	3

→ answer

i = 1 → S1 → from 0 to 0 ⇒ a
j = 3 → S2 → from 0 to 2 ⇒ aeb

```
int lcs( String s1, String s2) {
    int[][] dp = new int [N+1][M+1];
    NOTE: initialise 0th row & 0th col by 0.

    for(int i=1; i<=N; i++){
        for(int 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) → try to optimise space by making 2*m array.

~~~~~~~~~~~~~~

Edit Distance

~~~~~~~~~~~~~~

Given two strings S1 and S2. Convert String S1 into S2 by using folowwing operations:

* Insert a char : Ci
* Delete a char : Cd
* Replace a char : Cr

Every operation is associated with some cost.

Find minimum cost for conversion.

$Ci = 2$, $Cd = 2$, $Cr = 3$

eg1. S1 → a c (insert b)

S2 → a b c

cost = 2

S1 → a b c d

S2 → a b e

(I) delete c + deleted + insert e

2 + 2 + 2 ⇒ 6

(II) delete c + replace d

2 + 3 ⇒ (5) min Ans

S1 → a b ~~d~~ x ~~y~~ (c; insert g; delete)

S2 → a b c g x

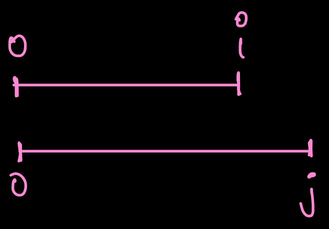
1 replace + 1 insertion + 1 delete

3 + 2 + 2

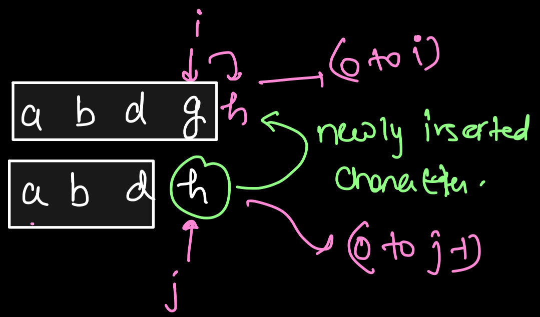
⇒ 7 Ans

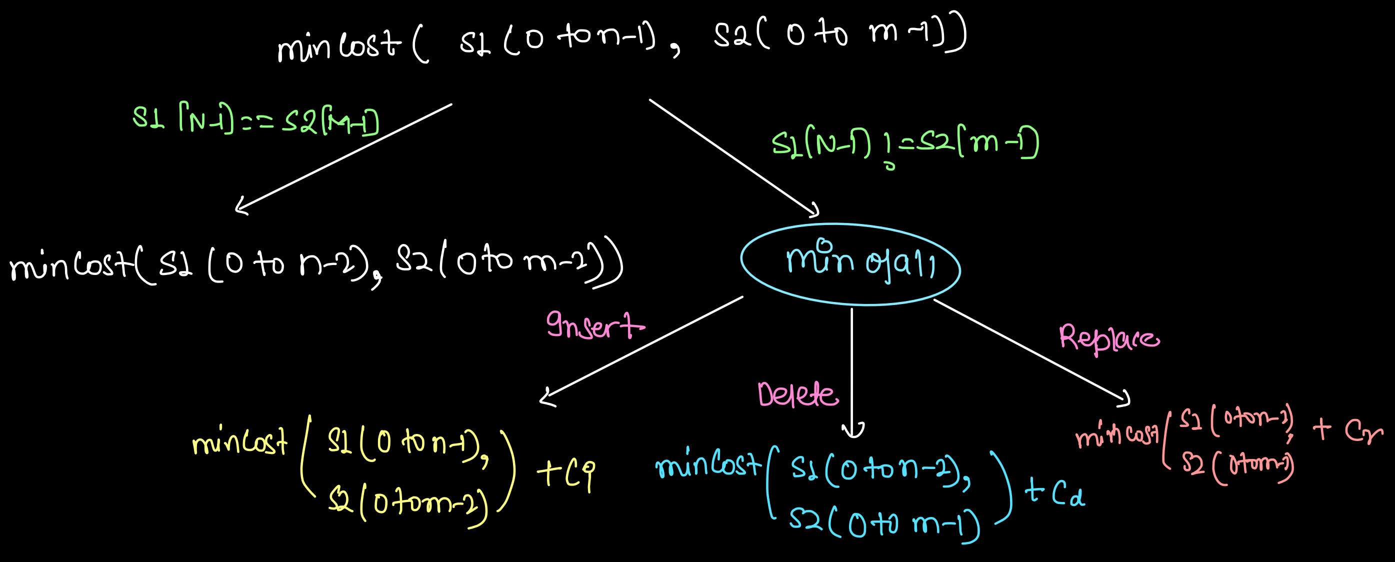
S1 → 0 to i

S2 → 0 to j

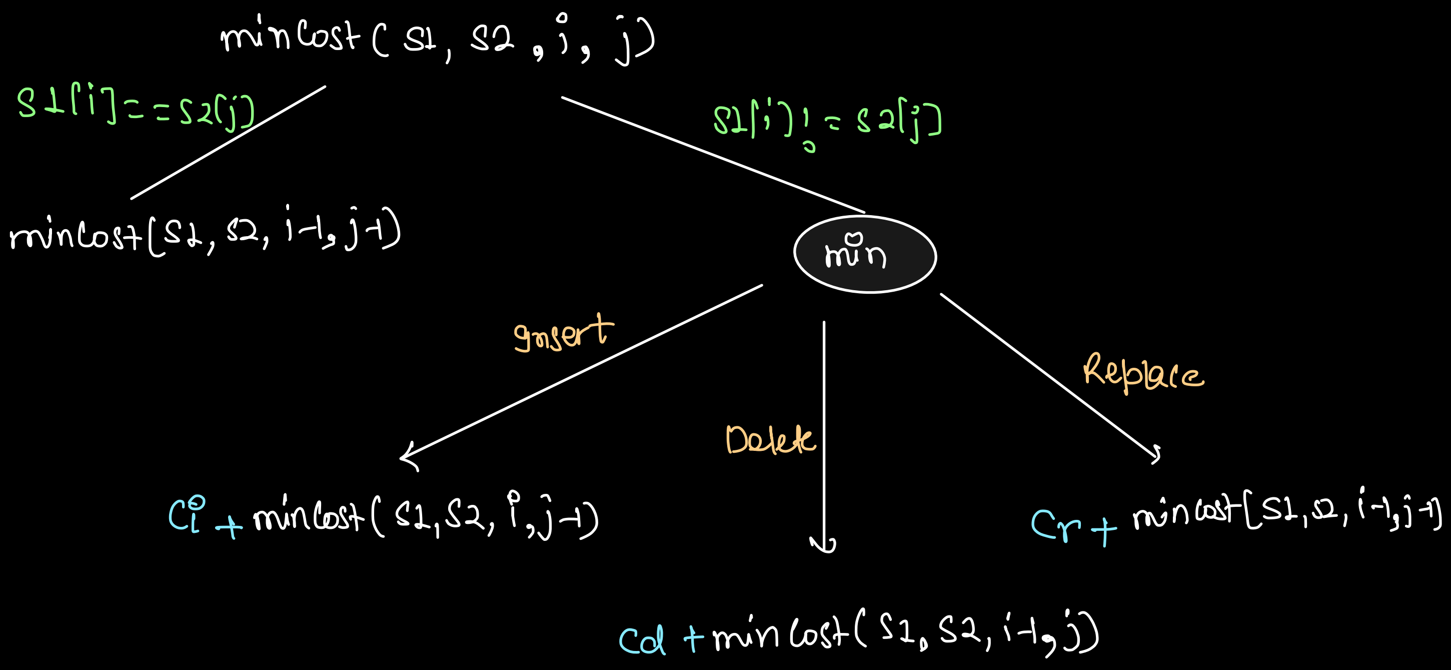


eg:





mincost(s1(0 to n-1), s2(0 to m-1))
s1[N-1] == s2[M-1]
s1(N-1) != s2(m-1)
mincost(s1(0 to n-2), s2(0 to m-2))
min of all
Insert
Delete
Replace
mincost(s1(0 to n-1), s2(0 to m-2)) + Ci
mincost(s1(0 to n-2), s2(0 to m-1)) + Cd
mincost(s1(0 to n-2), s2(0 to m-2)) + Cr


mincost(s1, s2, i, j)
s1[i]==s2[j]
s1[i]!=s2[j]
mincost(s1, s2, i-1, j-1)
min
Insert
Delete
Replace
ci + mincost(s1, s2, i, j-1)
cr + mincost[s1, s2, i-1, j-1]
cd + mincost(s1, s2, i-1, j)

#top down Approach:

```
int dp[N][m];
  → initialise with -1;
int mincost( s1, s2, i, j ) {
    if(i<0 && j<0) { return 0;}
    else if( i<0) { return ci*(j+1);} → eg: s1 → ab
                                            s2 → bcdab
    else if( j<0) { return cd*(i+1);}  eg s1 → abcde
                                          s2 →    de
     if( dp[i][j] != -1){
         return dp[i][j];
     }
     if( s1[i] == s2[j]) {
         dp[i][j]= mincost ( s1, s2, i-1, j-1);
     } else {
                        ci+ mincost(s1, s2, i, j-1);  → insert
         dp[i][j]= min { cd + mincost( s1, s2, i-1, j); → delete
                        cr + mincost( s1, s2, i-1, j-1); → replace
     }
     return dp[i][j];
}
```

T.C: O(n*m)

S.C. O(n*m)

#bottom up Approach:

			_	a	b	c
				0	1	2
			0	1	2	3
_		0	0	2	4	6
a	0	1	2			
b	1	2	4			
c	2	3	6			
d	3	4	8			

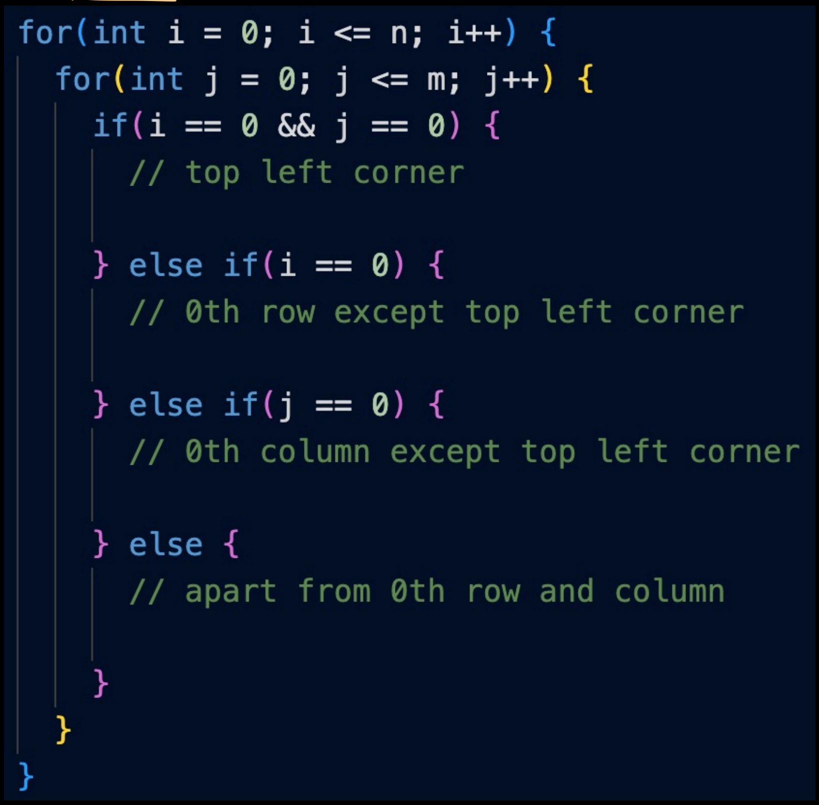
→ insertion

Ci → 2
Cd → 2
Cr → 3

$$dp[i][j] \quad i,j \begin{cases} s1[i] == s2[j] \to dp[i-1][j-1] \\ s1[i] \,!= s2[j] \end{cases}$$

$$\min \begin{cases} dp[i-1][j] + c \\ dp[i][j-1] + c \\ dp[i-1][j-1] + c \end{cases}$$

Structure :



```
for(int i = 0; i <= n; i++) {
    for(int j = 0; j <= m; j++) {
        if(i == 0 && j == 0) {
            // top left corner

        } else if(i == 0) {
            // 0th row except top left corner

        } else if(j == 0) {
            // 0th column except top left corner

        } else {
            // apart from 0th row and column

        }
    }
}
```

TODO: Code.
TC & SC

$\underbrace{\text{10:37 - 10:47 pm}}_{\text{Break}}$

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Wildcard Pattern Matching

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Given two strings S1 and S2. Check if they are matching or not.
S2 can contains '?' and '*', where
'?' -> it can match with any single character,
'*' -> it can match with 0 or more characters.

1. S1 → a b a c d
   S2 → a b a c d } true

2. S1 → a b a c d
   S2 → a ? a c ? } true
   (b above the first ?, d above the second ?)

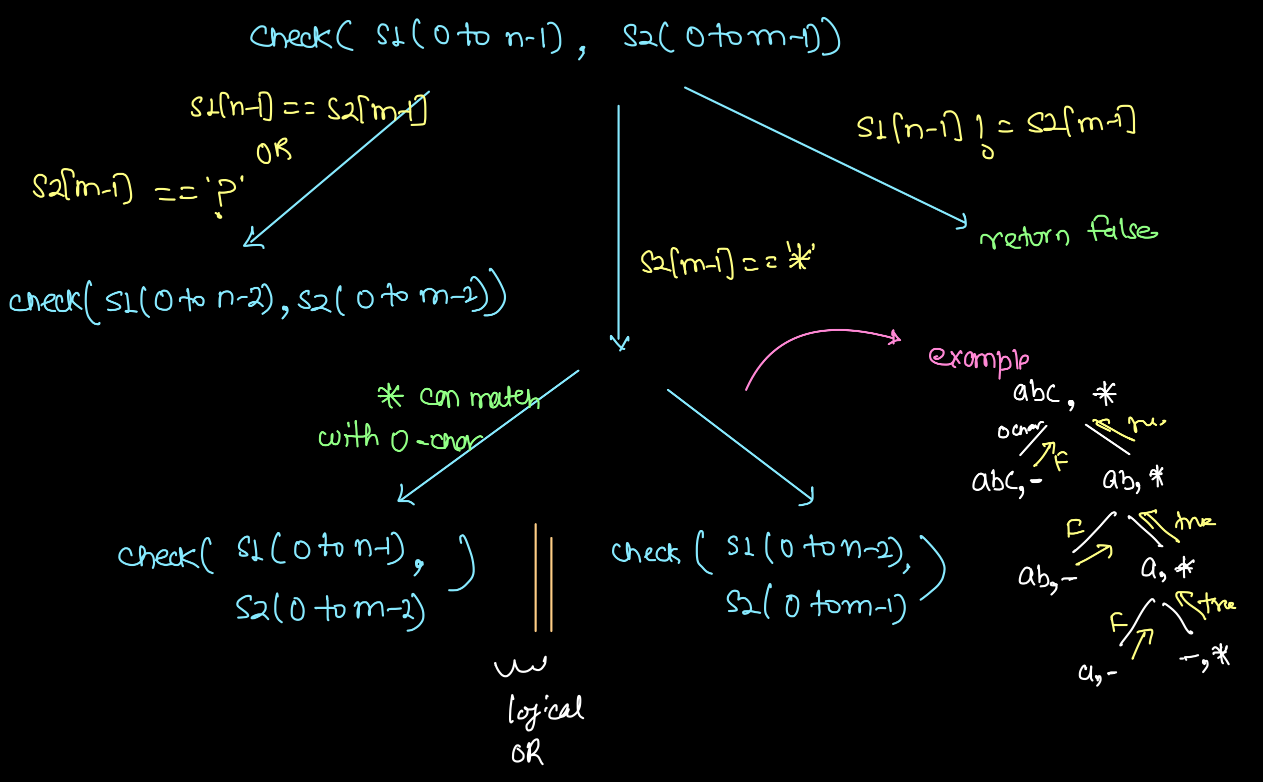
3. S1 → x b b z z c
   S2 → x * z * } true

4. S1 → x b b z z
   S2 → x * z * * * ? z } false

5. S1 → x b b z z
   S2 → x * z * * * ? } true.

S1 → char

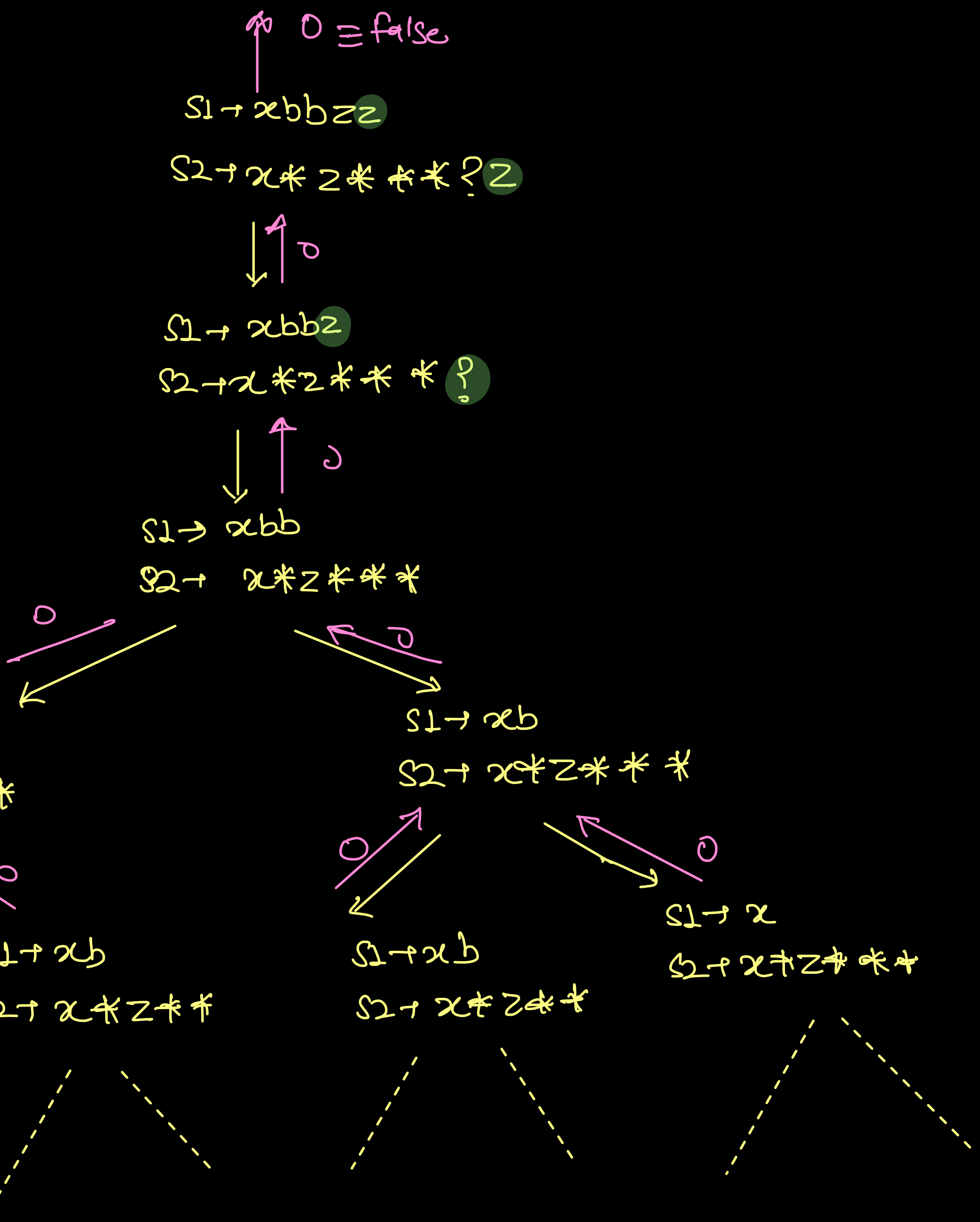
S2 → * → 0 char.
        → or more than 0 char.



Dry Run:

S1 → xbbzz

S2 → x*z***?z



0 → False

1 → true

#top- down Approah:

```
int dp[n][m];
⟶ initialise it  with -1;

int  check (s1, s2, i , j ) {                          s1 →   abc
                                                   eg: s2 → **abc
   if( i<0 && j<0){ return  1;}
   else if( i<0  && checkStar(s2,j)==true) {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);
     }
     else if( s2[j] == '*') {
                        check(s1, s2, i-1, j); ⟶ * with more char
        dp[i][j]=max {
                        check(s1, s2, i, j-1); → * with 0 char
     }
     else {
        dp[i][j]= 0;
     }                                        T.C: O(n*m)
                                              S.C: O(n*m)
   return dp[i][j];
}
booleam checkStar ( String S2 , int j)
    for(int i=0; i<=j; i++){
        if( s2[i] != '*'){
            return false;
        }
    }
    return true;
}
```

Bottom up

S1 → x b b z z c d

S2 → x * ? * d

boolem dp →

S2

		_	x	*	?	*	d
		0	1	2	3	4	5
_	0	T	F	F	F	F	F
x	1	F	T	T	F	F	F
b	2	F	F	T	T	T	F
b	3	F	F	T	T	T	F
z	4	F	F	T	T	T	F
z	5	F	F	T	T	T	F
e	6	F	F	T	T	T	F
d	7	F	F	T	T	T	T

S1

→ Be careful

→ true one.

i,j

s1[i] == s2[j] } i-1, j-1
s2[j] == '?' }

s2[j] == '*' (i-1,j) || (i,j-1)

s1[i] != s2[j] } 0 false

T.C: O(n*m)

SC: O(n*m)

TODO: Coding

NOTE: Be careful with 0th Row

Addition: Regular Expression matching