D.P with strings

- L.C.S (Longest Common Subsequence)
- Edit Distance
- Wildcard Pattern matching

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 , 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.

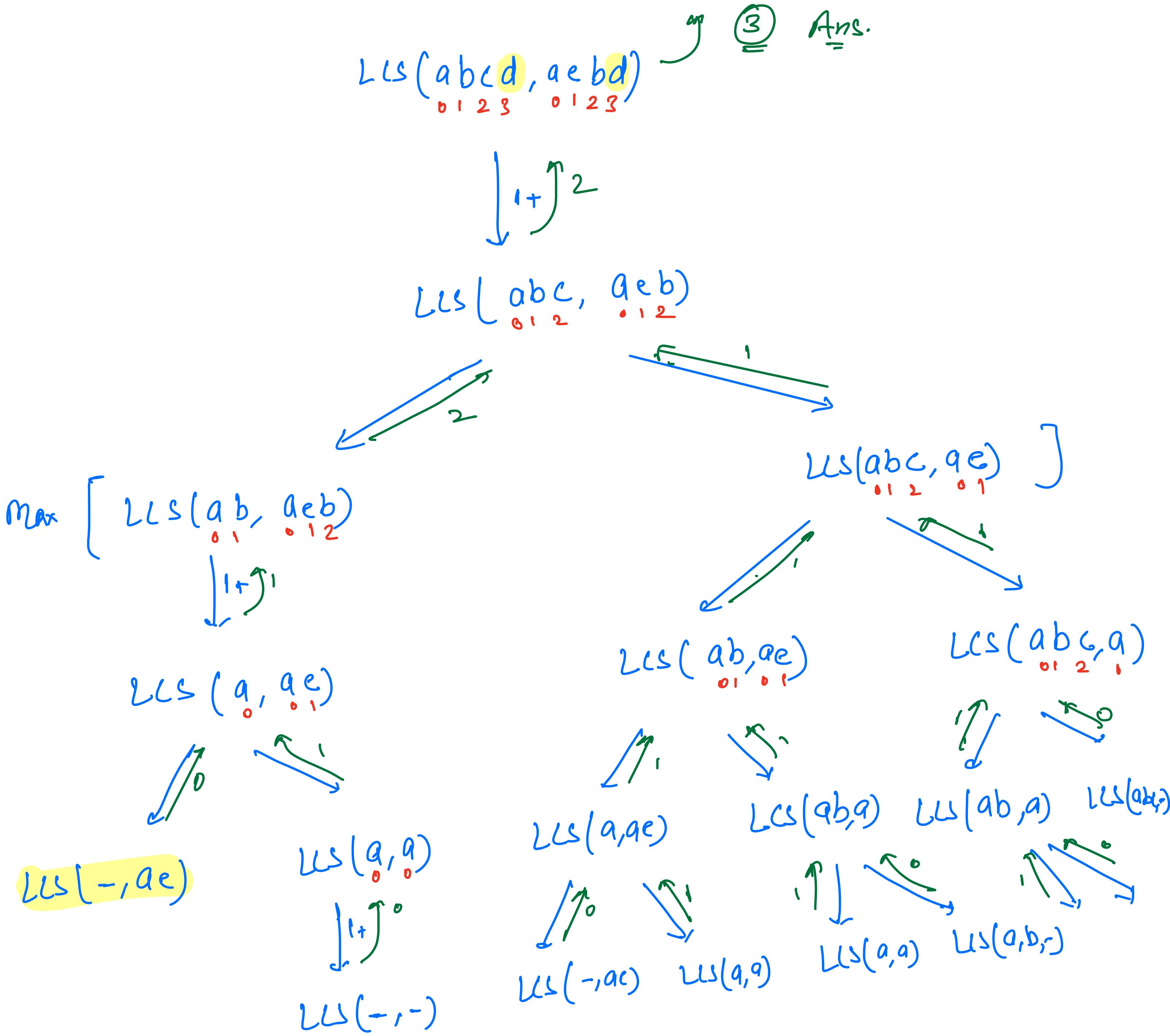
B.F idea. → Consider all subsequences of s1 and s2 & then find the longest common subsequence.

T.C → $O(2^N + 2^m + 2^N \cdot N)$

LCS (s1(0, N-1), s2(0, m-1))

s1[N-1] == s2[m-1] → 1 + LCS(s1(0,N-2), s2(0,m-2))

s1[N-1] != s2[m-1] → max { LCS(s1(0, N-2), s2(0,m-1)) ; LCS(s1(0, N-1), s2(0,m-2)) }



optimal sub-structure → ✓

overlapping subproblems → ✓

} ⇒ D.P

code:- **Top-down Approach**

dp[N][m], $\forall i,j$ dp[i][j] = -1;

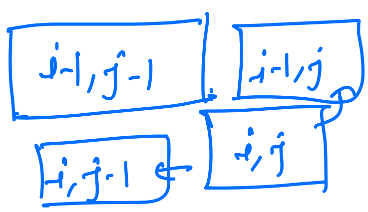
```
                  n-1  m-1
                   ↑    ↑
int lcs( s1, s2,   i ,  j , int dp[N][m] ){

    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)

s1 [0, i] $\xrightarrow{s1[i] == s2[j]}$ dp[i-1][j-1]

s2 [0, j] $\xrightarrow{s1[i] != s2[j]}$ Max(dp[i-1][j], dp[i][j-1])

dp[i][j]



s1 → abcd

s2 → aebd

dp[N+1][M+1]

		-	a	e	b	d	
			0	1	2	3	$\rightarrow j$
		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	← ans.
$\downarrow i$							

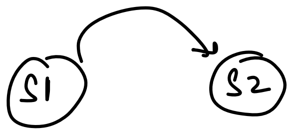
#code :-

```
int 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] = 1+ dp[i-1][j-1];
        }
        else{
            dp[i][j] = max( dp[i-1][j], dp[i][j-1]);
        }
    }
}
return dp[N][m];
```

$\boxed{\begin{array}{l} T.C \rightarrow O(N \times m) \\ S.C \rightarrow O(N \times m) \end{array}}$

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

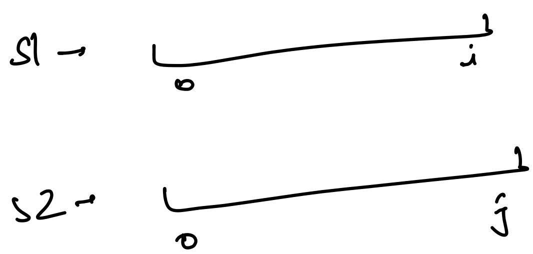
ans = 2.

s1 → a b ~~c~~ ~~d~~ (e)
s2 → a b e

1d + 1r
2+3 = 5.

s1 → a b ~~d~~ x ~~y~~ (c, g)
s2 → a b c g x

1 replacement + 1 insertion + 1 deletion
3+ 2+ 2 = 7.

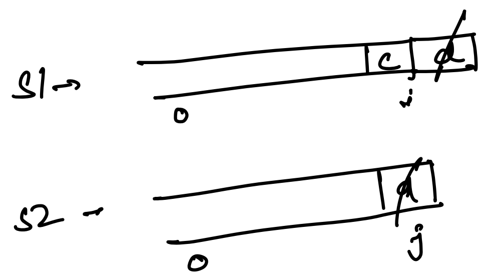


minCost(s1(0, N-1), s2(0, M-1))

$s1[N-1] == s2[M-1]$ → minCost(s1(0, N-2), s2(0, M-2))

$s1[N-1] \,!= s2[M-1]$ → min

- insert → C_i + minCost(s1(0, N-1), s2(0, M-2))
- delete → C_d + minCost(s1(0, N-2), s2(0, M-1))
- replace → C_r + minCost(s1(0, N-2), s2(0, M-2))



s1, s2, i, j

$s1[i] == s2[j]$ → s1, s2, i-1, j-1

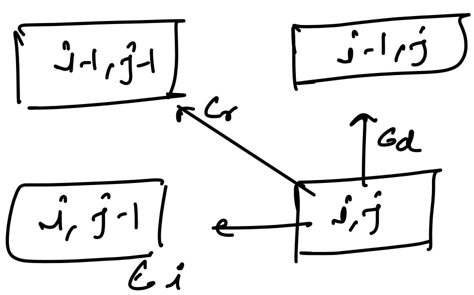
→ min

- insert → C_i + (s1, s2, i, j-1)
- delete → C_d + (s1, s2, i-1, j)
- replace → C_r + (s1, s2, i-1, j-1)

code → top-down

```
                                  n-1   m-1
                                   ↑     ↑
int minCost(s1, s2, i, j, int dp[n][m]){
        if(i<0 && j<0){ return 0}
        else if(i<0){ return Ci x (j+1); }
        else if(j<0){ return Cd x (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{
                          ⎧ Ci + minCost(s1, s2, i, j-1, dp);
            dp[i][j]=min  ⎨ 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 x m)
S.C → O(n x m)



bottom-up:

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

$C_i \to 2$

$C_d \to 2$

$C_r \to 3$

code → todo

int dp[N+1][m+1]

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

s2 – a b a c d

true.

② s1 → a b a c d

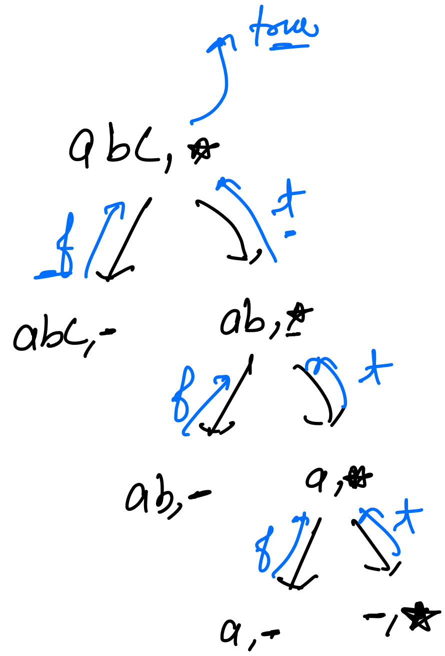
s2 → a ? a c ?

true.

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.

check(s1(0, N-1), s2(0, m-1))

s1[N-1] == s2[m-1]
or
s2[m-1] == '?'

check(s1(0, N-2), s2(0, m-2))

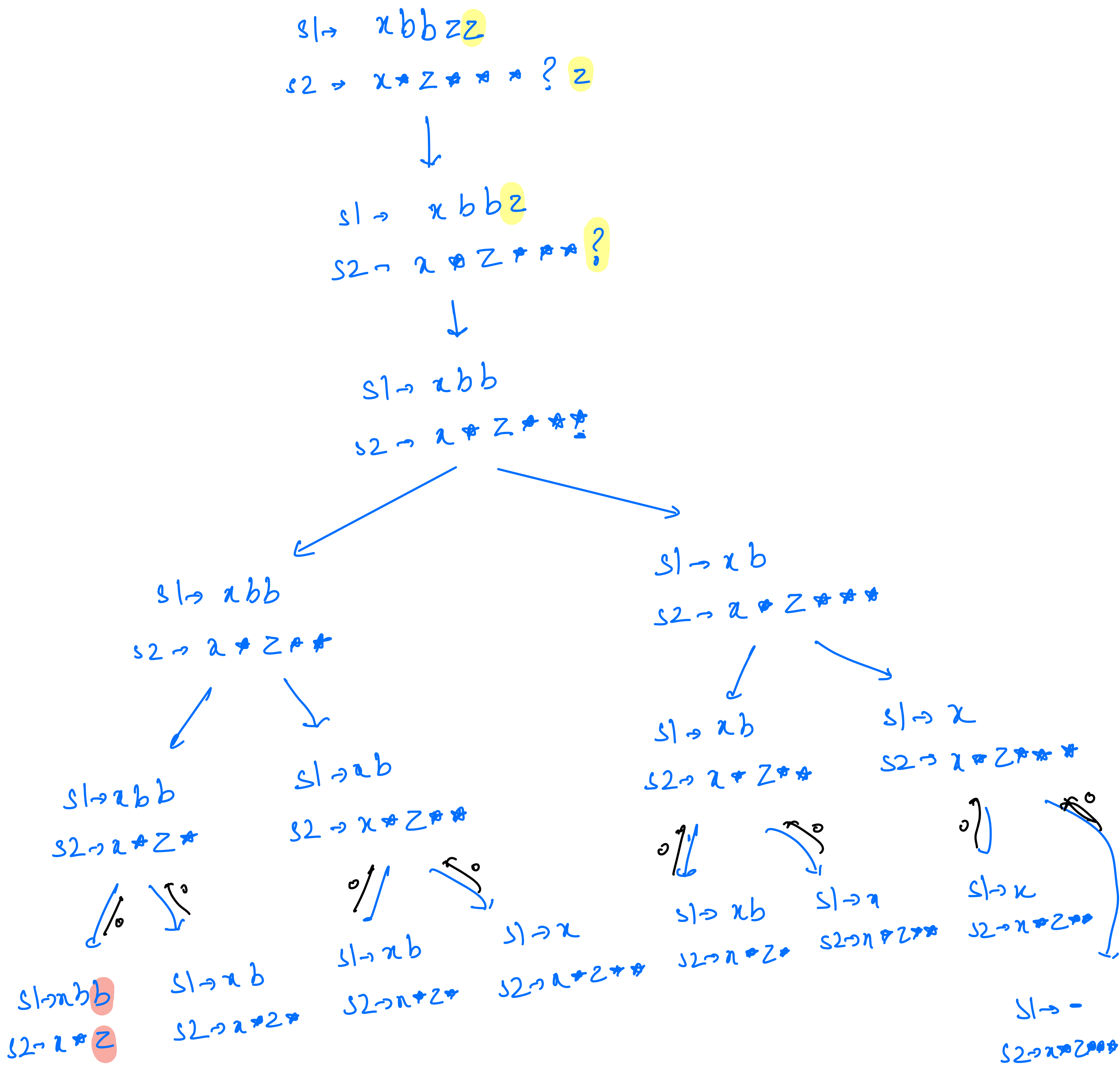
s2[m-1] == '*'

* matches
with 0 characters

check(s1(0, N-1), s2(0, m-2)) || check(s1(0, N-2), s2(0, m-1))

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

return false



s1 → xbbzz
s2 → x*z***?z
s1 → xbbz
s2 → x*z***?
s1 → xbb
s2 → x*z***
s1 → xbb
s2 → x*z**
s1 → xb
s2 → x*z***
s1 → xbb
s2 → x*z*
s1 → xb
s2 → x*z**
s1 → xb
s2 → x*z**
s1 → x
s2 → x*z***
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***

top-down → code.

```
int dp[N][m] , ∀ i,j → dp[i][j] = -1

                                   N-1   m-1
                                    ↑     ↑
int check ( s1, s2, i, j, dp[N][m]) {

    if (i<0 && j<0) { return 1 }
    else if ( i<0 && checkStars(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, dp);
    }
    else if ( s2[j] == '*') {
        dp[i][j] = max( check( s1, s2, i-1, j, dp);
                        check( s1, s2, i, j-1, dp);
    }
    else{
        dp[i][j] = 0;
    }
    return dp[i][j];
}
```

T.C → O(N×m)

S.C → O(N×m)

```
boolean checkStars( 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

x	*	?	*	d
0	1	2	3	4

			0	1	2	3	4	5
		0	t	f	f	f	f	f
x	0	1	f	t	t	f	f	f
b	1	2	f	f	t	t	t	f
b	2	3	f	f	t	t	t	f
z	3	4	f	f	t	t	t	f
z	4	5	f	f	t	t	t	f
c	5	6	f	f	t	t	t	f
d	6	7	f	f	t	t	t	t

ans.

code - todo.

Regular Expression Matching