

D.P with strings

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

Longest Common Subsequence \rightarrow (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

a c d g f, b c d g f

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. \rightarrow Consider all subsequences of s1 and s2 & then find the longest common subsequence. T.C $\rightarrow O(2^N + 2^m + 2^N \cdot N)$

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

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



$$1 + LCS(s1(0, N-2), s2(0, m-2))$$

$$s1[N-1] \neq s2[m-1]$$



$$\max \begin{cases} LCS(s1(0, N-2), s2(0, m-1)) \\ LCS(s1(0, N-1), s2(0, m-2)) \end{cases}$$

$LCS(abcd, aebd)$ \rightarrow 3 Ans.

0 1 2 3 0 1 2 3

$\downarrow \uparrow^2$

$LCS(abc, aeb)$

0 1 2 0 1 2

$\swarrow \searrow^2$

$\swarrow \searrow^1$

$LCS(abc, ae)$

0 1 2 0 1

$\max [LCS(ab, aeb)]$

0 1 0 1 2

$\downarrow \uparrow^1$

$LCS(a, ae)$

0 0 1

$\swarrow \searrow^0$

$\swarrow \searrow^1$

$LCS(a, a)$

0 0

$\downarrow \uparrow^0$

$LCS(-, -)$

$LCS(-, ae)$

$LCS(ab, ae)$

0 1 0 1

$\swarrow \searrow^1$

$LCS(a, ae)$

$\swarrow \searrow^0$

$LCS(-, ae)$

$\swarrow \searrow^1$

$LCS(a, a)$

$\swarrow \searrow^1$

$LCS(ab, a)$

$\swarrow \searrow^0$

$LCS(a, a)$

$\swarrow \searrow^0$

$LCS(a, b)$

$\swarrow \searrow^0$

$\swarrow \searrow^0$

$LCS(abc, a)$

0 1 2 1

$\swarrow \searrow^1$

optimal substructure \rightarrow ✓

overlapping subproblems \rightarrow ✓

} \Rightarrow D.P

code. → Top-down Approach

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

int lcs (s1, s2, ⁿ⁻¹ i, ^{m-1} 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);

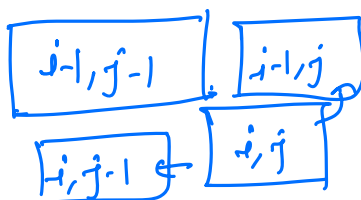
else {

dp[i][j] = Max (lcs (s1, s2, i-1, j, dp), lcs (s1, s2, i, j-1, dp));

return dp[i][j];

T.C → $O(N \times m)$
S.C → $O(N \times m)$

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



s1 → a b c d

$dp[n+1][m+1]$

s2 → a e b d

Diagram illustrating the DP table for the Longest Common Subsequence (LCS) problem between strings s1 = "abcd" and s2 = "aebd".

The DP table is a 5x5 grid (indices 0 to 4). The first row and first column are initialized to 0.

The table values are:

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	1	1	1
2	0	1	1	2	2
3	0	1	1	2	2
4	0	1	1	2	3

The value 3 in the cell (4, 4) is highlighted in green and labeled as the answer (ans.).

#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] );
```

```
            } {
```

```
        }
```

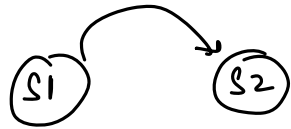
```
    }
```

```
return dp[n][m];
```

T.C → $O(N \times m)$
S.C → $O(N \times m)$

Edit - Distance

Given $s1$ & $s2$. Convert $s1 \rightarrow s2$ by using some operations in $s1$ only.



- ① insert $\rightarrow C_i$
- ② delete $\rightarrow C_d$
- ③ replace $\rightarrow C_r$

find minimum cost to convert $s1$ to $s2$.

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

Ex \rightarrow $s1 \rightarrow ac$
 $s2 \rightarrow abc$

ans = 2.

$s1 \rightarrow ab \overset{e}{\cancel{d}}$ $1d + 1i$
 $s2 \rightarrow abe$ $2 + 3 = \underline{5}$

$s1 \rightarrow ab \overset{c}{\cancel{d}} \overset{g}{\cancel{y}}$
 $s2 \rightarrow abcgx$

1 replacement + 1 insertion + 1 deletion

$$3 + 2 + 2 = \underline{7}$$

$s1 \rightarrow$

$s2 \rightarrow$

$$\text{minlast}(s1(0, N-1), s2(0, m-1))$$

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

$$s1[N-1] \neq s2[m-1]$$

$$\text{minlast}(s1(0, N-2), s2(0, m-2))$$

min

insert

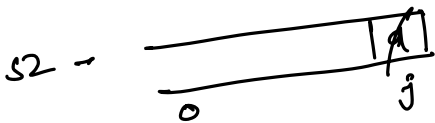
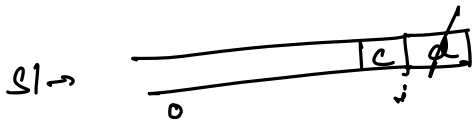
delete

replace

$$C_i + \text{minlast}(s1(0, N-1), s2(0, m-2))$$

$$C_d + \text{minlast}(s1(0, N-2), s2(0, m-1))$$

$$C_r + \text{minlast}(s1(0, N-2), s2(0, m-2))$$



s1, s2, i, j

$$s1[i] == s2[j]$$

min

insert

delete

replace

s1, s2, i-1, j-1

$$C_i + (s1, s2, i, j-1)$$

$$C_d + (s1, s2, i-1, j)$$

$$C_r + (s1, s2, i-1, j-1)$$

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 * (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 );
            Cx + minCost ( s1, s2, i-1, j-1, dp );
        }

```

```

    }

```

```

    return dp[i][j];

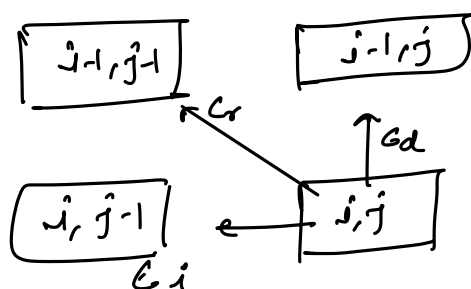
```

```

}

```

$T.C \rightarrow O(n \times m)$
 $S.C \rightarrow O(n \times m)$



bottom-up:

Handwritten diagram illustrating the construction of a 2D array for the Floyd-Warshall algorithm. The array is a 5x5 grid. The columns are labeled 0, 1, 2, 3, 4. The rows are labeled 0, 1, 2, 3, 4. The diagonal elements (0,0), (1,1), (2,2), (3,3), (4,4) are all 0. The elements (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4) are all 1. The elements (1,0), (2,0), (3,0), (4,0), (2,1), (3,1), (4,1), (3,2), (4,2), (4,3) are all 1. The elements (0,0), (1,1), (2,2), (3,3), (4,4) are all 0. The elements (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4) are all 1. The elements (1,0), (2,0), (3,0), (4,0), (2,1), (3,1), (4,1), (3,2), (4,2), (4,3) are all 1.

$$C_i \rightarrow 2$$
$$Cd \Rightarrow 2$$
$$L_r \rightarrow 3$$

code - todo

int dp[N+1][M+1]

Wildcard Pattern Matching

Given $s1$ & $s2$. Check if they are matching.

$s2 \rightarrow$ it can contain '?', '*'

\downarrow matches with any single character.

\rightarrow matches with 0 or more characters.

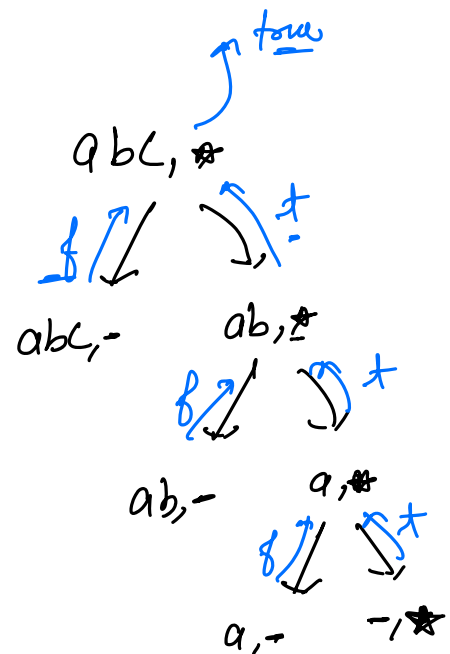
① $s1 \rightarrow a b a c d$
 $s2 \rightarrow a b a c d$ true.

② $s1 \rightarrow a b a c d$
 $s2 \rightarrow a ? a c ?$ true.

③ $s1 \rightarrow \underline{x} \underline{b} \underline{b} \underline{z} \underline{z} \underline{c}$
 $s2 \rightarrow \underline{x} \underline{*} \underline{z} \underline{*}$ true.

④ $s1 \rightarrow \underline{x} \underline{b} \underline{b} \underline{z} \underline{z}$
 $s2 \rightarrow \underline{x} \underline{*} \underline{z} \underline{*} \underline{*} \underline{*} \underline{?} \underline{z}$ false.

⑤ $s1 \rightarrow \underline{x} \underline{b} \underline{b} \underline{z} \underline{z}$
 $s2 \rightarrow \underline{x} \underline{*} \underline{z} \underline{*} \underline{*} \underline{*} \underline{?}$ 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

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

return false

$check(s1(0, N-1), s2(0, m-2)) \parallel check(s1(0, N-2), s2(0, m-1))$

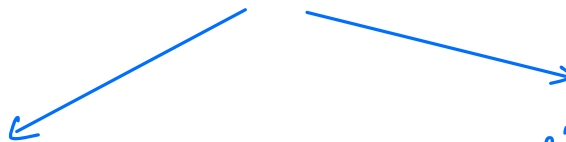
$s1 \rightarrow xbbzz$
 $s2 \rightarrow x * z * * * ? z$



$s1 \rightarrow xbbz$
 $s2 \rightarrow x * z * * * ?$



$s1 \rightarrow xbb$
 $s2 \rightarrow x * z * * *$

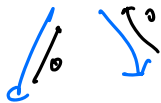


$s1 \rightarrow xbb$
 $s2 \rightarrow x * z * *$



$s1 \rightarrow xbb$
 $s2 \rightarrow x * z *$

$s1 \rightarrow xbb$
 $s2 \rightarrow x * z *$



$s1 \rightarrow xbb$
 $s2 \rightarrow x * z *$

$s1 \rightarrow xb$
 $s2 \rightarrow x * z * * *$



$s1 \rightarrow xb$
 $s2 \rightarrow x * z *$

$s1 \rightarrow x$
 $s2 \rightarrow x * z * * *$



$s1 \rightarrow xb$
 $s2 \rightarrow x * z *$

$s1 \rightarrow x$
 $s2 \rightarrow x * z *$

$s1 \rightarrow x$
 $s2 \rightarrow x * z *$

$s1 \rightarrow -$
 $s2 \rightarrow x * z *$

top-down → code.

int dp[n][m], $\forall i, j \rightarrow dp[i][j] = -1$

int check (s1, s2, $\overset{n-1}{\uparrow} i, \overset{m-1}{\uparrow} j, dp[n][m])$

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 ($dp[i][j] \neq -1$) { 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]; }

$\left[\begin{array}{l} T.C \rightarrow O(N \times m) \\ S.C \rightarrow O(N \times m) \end{array} \right]$

```
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	x	f	f	f	f	f
1	f	x	x	f	f	f
2	f	f	x	x	x	f
3	f	f	x	x	x	f
4	f	f	x	x	x	f
5	f	f	x	x	x	f
6	f	f	x	x	x	f
7	f	f	x	x	x	x

code → todo.

ans.

Regular Expression Matching