

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 or 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 longest common subsequence.

$$T.C \rightarrow O(2^n + 2^m + 2^{n \times m})$$

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

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

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

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

$$\text{Max} \left[\begin{array}{l} L.C.S(s1(0, N-2), s2(0, M-1)), \\ L.C.S(s1(0, N-1), s2(0, M-2)) \end{array} \right]$$

$$\text{LCS}(abcd, aebd) \rightarrow \text{ans. } 3$$

0 1 2 3 0 1 2 3

$$\downarrow \uparrow + 2$$

$$\text{LCS}(abc, aeb)$$

0 1 2 0 1 2

$$\swarrow \searrow + 2$$

$$\swarrow \searrow + 1$$

$$\text{Max} \left[\text{LCS}(ab, aeb) \quad \text{LCS}(abc, ae) \right]$$

0 1 0 1 2 0 1 2 0 1

$$\downarrow \uparrow + 1$$

$$\text{LCS}(a, ae)$$

0 0 1

$$\swarrow \searrow + 0$$

$$\swarrow \searrow + 1$$

$$\text{LCS}(-, ae)$$

0 1

$$\text{LCS}(a, a)$$

0 0

$$\downarrow \uparrow + 0$$

$$\text{LCS}(-, -)$$

$$\text{Max} \left[\text{LCS}(ab, ae) \right]$$

0 1 0 1

$$\text{LCS}(abc, a)$$

0 1 2 0

$$\swarrow \searrow + 0$$

$$\swarrow \searrow + 0$$

$$\text{LCS}(a, ae)$$

$$\text{LCS}(ab, a)$$

$$\swarrow \searrow + 0$$

$$\swarrow \searrow + 0$$

$$\text{LCS}(-, ae)$$

$$\text{LCS}(a, a)$$

$$\text{LCS}(a, a)$$

$$\downarrow \uparrow +$$

$$\downarrow \uparrow +$$

$$\text{LCS}(-, -)$$

$$\text{LCS}(-, -)$$

$$\swarrow \searrow + 0$$

$$\swarrow \searrow + 0$$

$$\text{LCS}(ab, a)$$

$$\text{LCS}(abc, -)$$

$$\swarrow \searrow + 0$$

$$\text{LCS}(ab, -)$$

$$\downarrow \uparrow +$$

$$\text{LCS}(-, -)$$

optimal substructure ✓

over-lapping sub-problems ✓

⇒ D.P ✓

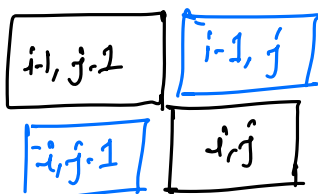
code. →

Top-Down Approach

```
int lcs( s1, s2, i, j, int dp[N][M]) {  
     $\begin{matrix} n-1 & m-1 \\ \downarrow & \downarrow \end{matrix}$   
    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 → $O(N \times m)$

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



s1 → a b c d

s2 → a e b d

- a | e | b | d
 0 1 2 3

		0	1	2	3	4
	0	0	0	0	0	0
a	1	0	1	1	1	1
b	2	0	1	1	2	2
c	3	0	1	1	2	2
d	4	0	1	1	2	3

→ ans.

#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]);

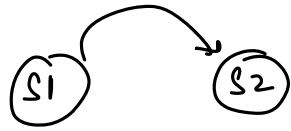
}

return dp[N][m];

T.C → O(N*m)
S.C → O(N*m)

Edit - Distance

Given $s1$ & $s2$. Convert $s1 \rightarrow s2$ 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

$s1 \rightarrow ab\cancel{c}$
 $s2 \rightarrow abc$

1 deletion +
1 replacement

ans = 5.

$s1 \rightarrow ab\cancel{c}xy$
 $s2 \rightarrow abcgx$

1 replacement + 1 deletion + 1 insertion

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

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

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

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

Min

insert

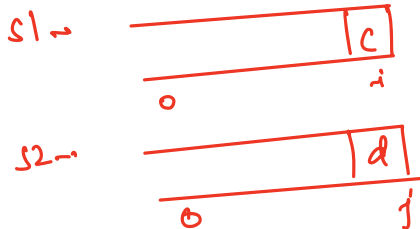
delete

replace

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

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

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



optimal substructure ✓
overlapping subproblems ✓

s1, s2, i, j

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

$$s1[i] \neq s2[j]$$

s1, s2, i-1, j-1

Min

$C_i +$

$C_d +$

C_r

s1, s2, i, j-1

s1, s2, i-1, j

s1, s2, i-1, j-1

code \rightarrow top-down

```

int minCost ( s1, s2, n-1i, m-1j , int dp[N][M]) {

```

```

    if (i < 0 && j < 0) { return 0; }
    else if (i < 0) { return C_i * (j+1); }
    else if (j < 0) { return C_d * (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 {
            C_i + minCost ( s1, s2, i, j-1, dp );
            C_d + minCost ( s1, s2, i-1, j, dp );
            C_x + minCost ( s1, s2, i-1, j-1, dp );
        }

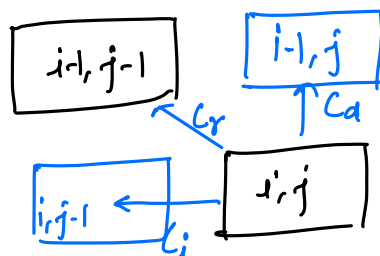
```

```

    return dp[i][j];
}

```

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



bottom-up:

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

↪ ans.

$C_i \rightarrow 2$

$C_d \rightarrow 2$

$C_r \rightarrow 3$

#code - todo.

Wildcard Pattern Matching

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

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

↓
matches with any
single character.

→ 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 $s1 \rightarrow x b b z z c$

$s2 \rightarrow x * z *$

true

④ $s1 \rightarrow x b b \underline{z} \underline{z}$

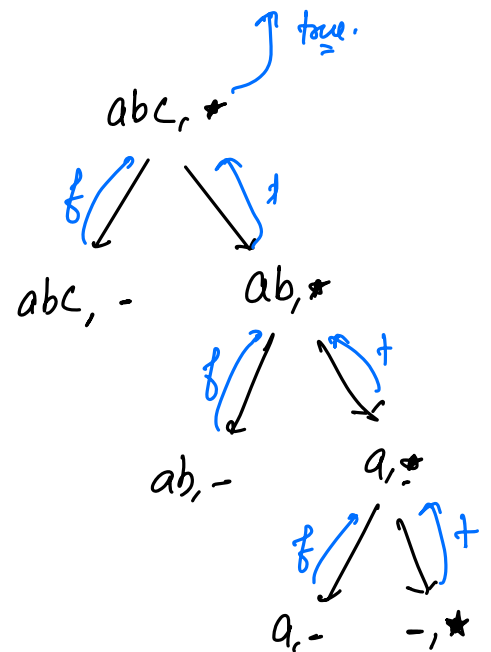
$s2 \rightarrow x * z * \underline{\quad} \underline{\quad} ? \underline{\quad}$

false.

⑤ $s1 \rightarrow \underline{x} \underline{b} \underline{b} \underline{z} \underline{z}$

$s2 \rightarrow \underline{x} \underline{*} \underline{z} \underline{*} \underline{*} \underline{*} ?$

true.



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

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

OK

$s2[m-1] == '?'$

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

$s2[m-1] == '*'$

* matches
with 0 chars

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

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

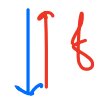
$s1[N-1] != s2[m-1]$

return false.

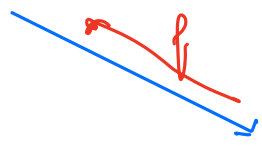
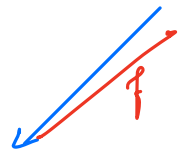
$s1 \rightarrow xbbzz$
 $s2 \rightarrow x * z * * * ? z$



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

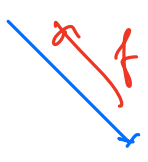
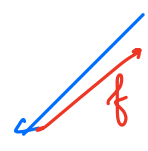


$s1 \rightarrow xbb$
 $s2 \rightarrow x * z * * *$



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

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

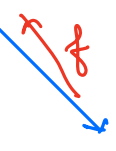
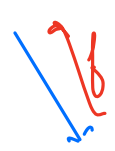
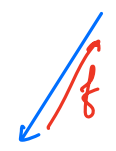


$s1 \rightarrow xbb$
 $s2 \rightarrow x * z *$

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

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

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



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

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

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

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

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

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

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

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



$t \rightarrow 1$
 $f \rightarrow 0$

optimal sub-structure ✓
 overlapping sub-problems ✓

$\rightarrow D.P$

top-down → code.

```
int check (s1, s2, i, j, dp[N][M]){  
    if ( i < 0 && j < 0 ) { return 1 }  
    else if ( i < 0 && checkStart(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 → $O(N \times M)$
S.C → $O(N \times M)$

```
boolean checkStars(s2, j){
```

```
    for(i=0; i ≤ j; i++){
```

```
        if(s2[j] != '*') { return false; }
```

```
    }
```

```
    return true;
```

x

x

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

ans.

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.

$$arr[i]$$

Sum.

$$(2, 3, 5, 7)$$

→

100

