DP on strings:-

① Longest common subsequence:- → order same.

s1 = "adebc" s2 = "dcadb".

No. of subsequence = 2^n → length of the string

adebc

a ✓
d
e
b
c
ad
:

dcadb.

d
c
a ✓
d
b
dc
:

Find common among both the subsequences of the string, and the one with longest length will be our ans.

"adb" → length 3

Brute Force → Generate All subsequences and compare linearly → Exponential TC.

Rules to write recurrence:-

- Express in terms of index.
- Explore all possibilities on that index.
- Take best among them.

Since we have 2 strings express them in terms of ind1 and ind2.

f(2, 2) → lcs of string 1 from (0...2) & str2 from (0...2)

acd | ced
↑ ↑

d, d matching i.e we got a lcs of length 1.

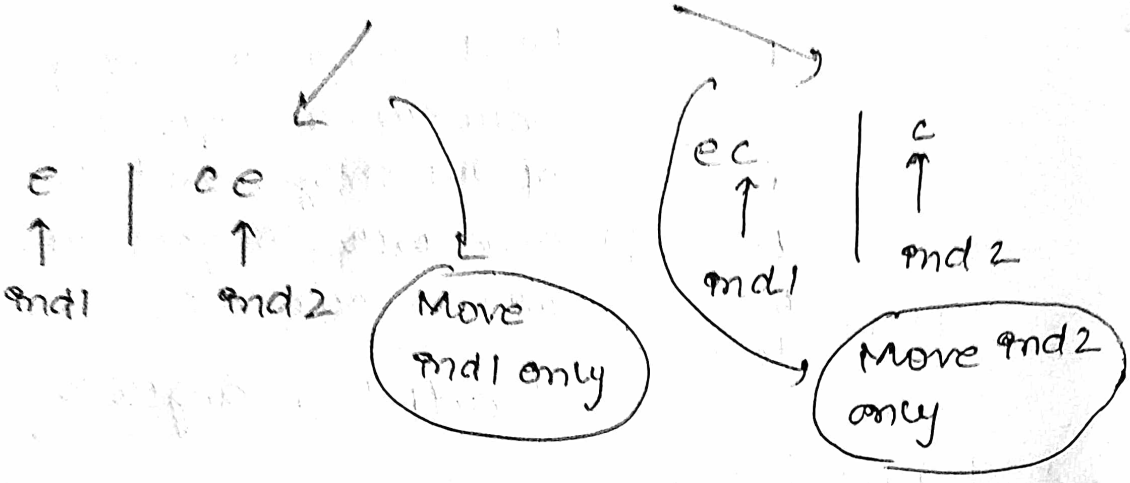
then we shrink the string, 1 + ac/ce

If (s1[ind] == s2[ind]), 1 + f(ind-1, ind2-1)

↓ Recurrence if both are matching.

Not Matching → a c | c e

ind1 ind2



If not match:-

$$0 + \{f(ind-1, ind2), f(ind1, ind2-1)\}$$

↓ Max of both

Base Condn:-

If (ind1 < 0 || ind2 < 0) {

return 0

}

If Right shift ends. ind1 == 0 || ind2 == 0

This means that we reached the end of the string.

Recursion → $(2^n \times 2^m)$ → exponential.

Overlapping sub-problems.

Apply DP.

f(i,j) → lcs of a str1[0...i] & str2[0...j]

N M DP(N×M)

TC: O(N×M) SC: O(N×M) + O(N+M)

Memoization ASS

Bottom Up:-

- Copy the base case
- Write changing para in opp fashion.
- Copy the recurrence.

Code:-

```
f lcs(string s, string t, int ind1, int ind2, dp)
    if(ind1==0 || ind2==0) return 0;
    if(dp[ind1][ind2] != -1) {
        return dp[ind1][ind2];
    }
    if(s[ind1-1] == t[ind2-1]){
return dp[ind1][ind2] = 1 + lcs(s, t, ind1-1, ind2-1, dp)
}
return dp[ind1][ind2] = max(lcs(s,t,ind1-1,ind2,dp),
                            lcs(s,t,ind1,ind2-1,dp))

main() {
    n = s.size(), m = t.size();
    dp(n+1, vec<int>(m+1, -1));
    lcs(s, t, n, m, dp)
```

Bottom-Up

```
int lcs (string s, string t) {
    n = s.size(), m = t.size();
vector<vector<int>> dp(n+1, vector<int>(m+1,-1));
for(j=0 <= m) dp[0][j] = 0;
for (i=0 <= n) dp[i][0] = 0;

for(int i=1; i<=n; i++) {
    for(int j=1; j<=m; j++) {
        if(s[i-1] == t[j-1])
            dp[i][j] = 1 + dp[i-1][j-1]
        else
            dp[i][j] = max(dp[i][j-1], dp[i-1][j])
    }
}
return dp[n][m];
```

Print LCS

	j →	0	1	2	3	4	5
i		0	0	0	0	0	0
			b	d	g	e	k
a	1	0	0	0	0	0	0
b	2	0	1	1	1	1	1
c	3	0	1	1	1	1	1
d	4	0	1	2	2	2	2
e	5	0	1	2	2	3	3

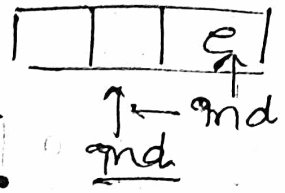
If char matches add the char present to string and move diagonally, else move to max of i-1, j & i, j-1

Code :

Same as LCS,

```
int len = dp[n][m]
string ans = "";
for(i=0; i< len; i++) {
    ans += '#';
}
index = len-1; i=n, j=m;
while(i>0 && j>0) {
    if(s[i-1] == t[j-1]) {
        ans[index] = s[i-1];
        index--;
        i--; j--;
    }
    else if(dp[i-1][j] > dp[i][j-1]) {
        i--;
    }
    else {
        j--;
    }
}
cout << ans;
```

✓ start storing index from end of the string.



TC: $O(N \times M) + O(N+M)$

Longest common substring

↓ consecutive.

s1 = "abcjklp" s2 = "acjkp" (L=3)

Here the order should be same,

From LCS, match cond^{n} is same,

s1[i-1] == s2[j-1], 1+dp[i-1][j-1]. But the not-match cond^{n} would change, we cannot shrink and look out for max cond^{n}.

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

If match 1+ prev

1+prev ↓ 0

In the else cond^{n} from LCS,

dp[i][j] = 0 (Only change)

Longest common substring is the max val in matrix.

```
ans = 0;
Everytime same,
    if (s[i-1] == t[j-1]) {
        dp[i][j] = 1 + dp[i-1][j-1]
        ans = max(ans, dp[i][j])
    }
    else {
        dp[i][j] = 0;
    }
} return ans;
```

Longest Palindrome Subsequence

s1 = "bbabcbcab" O/P: 7

s2 = "bacbebabb" reverse it & find longest common subsequence among both, that would be the longest palindromic subsequence.

Code:-

```
LCS (s,t)

main() {
    string s1 = s;
    reverse (s1.begin(), s1.end());
    return lcs(s, s1);
}
```

Min Insertions to make a string Palindrome.

s = "abcaa"

Max operations = length of the string

We can reverse the given string and concatenate it.

But here, we need to find min no. of operations.

Approach → Keep the longest palindrome section intact.

abcaa, LPS → a a a

a bc a c b a

reverse

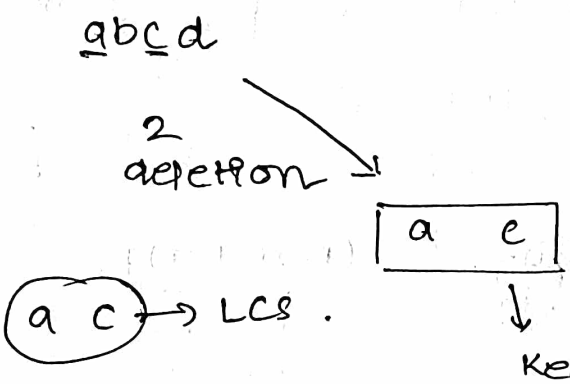
∴ Ans = n − LPS

Min Operations to convert str1 → str2.

→ Delete, Insert Anywhere.

str1 = "abcd" str2 → "anc"

Brute → Delete everything and insert same char. But we need min operations, now to do this.



∴ No. of Deletion = n − len(lcs)

No. of insertion = m − len(lcs)

Total Operations = n + m − 2 × len(LCS) → Answer.

Shortest common supersequence :-

s1 = "brute", s2 = "groot"

"brutegroot"
len = 10

means both of the strings are there in it. But is it the shortest?

"bgruootte" → length = 8.

↓

contains both "brute" and "groot".

Print length of the scs. & find scs.

s1 = "brute" s2 = "groot"

→ Take common characters only once.

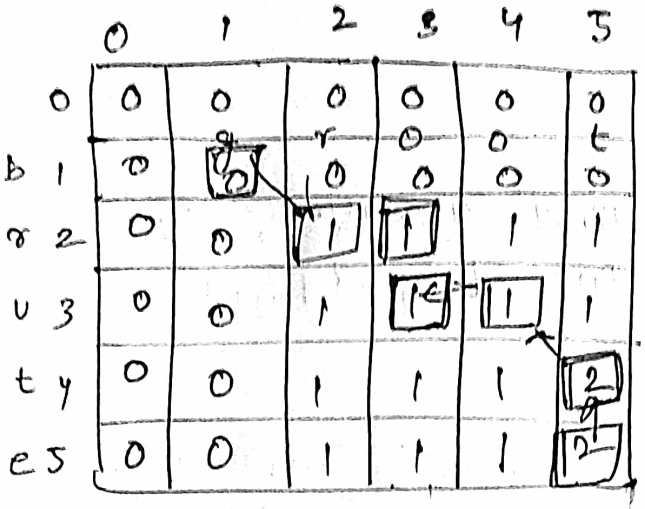
- brute (rt) groot

$$\therefore \boxed{SCS = n+m - len(lcs)}$$

$$= 5+5-2$$

$$= 8.$$

How to find & print the SCS.



string = etouorbg

↓ reverse

gbrouote.

LCS — Code

```
string ans = "";  i = n, j = m;
while (i > 0 && j > 0) {
    if (s[i-1] == t[j-1]) {
        ans += s[i-1]; i--; j--;
    }
    else if (dp[i-1][j] > dp[i][j-1]) {
        ans += s[i-1]; i--; }
    else {
        ans += t[j-1]; j--;
    }
while (i > 0) ans += s[i-1] i--;
while (j > 0) ans += t[j-1] j--;
reverse (ans)
return ans;
```

Distinct subsequences :-

s1 = "babgbag" s2 = "bag"
t

Return no. of distinct subsequences of s which equals t.

babgbag

babgbag
babgbag
babgbag
babgbag
babgbag

Total = 5

f(n-1, m-1) → No. of distinct subsequences of t[0...j] in s[0...i]
↓ ↓
i j

Base cases :-

i
↓ babgbag
-1 0

j
↓ bag
-1 0

When to return 1? → If all characters of t matched.

```
f (i,j) {
   if (j<0) return 1;
   if (i<0) return 0;
   if (s[i] == t[j]) {
      return ( f(i-1,j-1) + f(i-1,j));
   }
   else {
     return f(i-1,j);
   }
```

If 1 based
j==0
i==0
s[i-1] = t[j-1]

TC: $O(2^n \times 2^m)$ dp(n+1, . . m+1

SC: O(N+M)

Memoization :- (TLE)

i, j
↓ ↓
[N][M]

using 2D-vector, TC: O(NXM)

SC: O(NXM) + O(N+M)

Tabulation (j=0) → 1 (string matched) → return 1

```
f(s,t)
 vec<vec<double>> dp(n+1, vector<double>(m+1, 0))
 for(i=0; i<=n; i++) dp[i][0] = 1;
 for(i=1; i<=n; i++) {
   for(j=1 <=m){ if(s[i-1] == t[j-1]) {
       dp[i][j] = dp[i-1][j-1] + dp[i-1][j];
   }
   else {
      dp[i][j] = dp[i-1][j];
   }
 }
```

Edit Distance :-

s1 = "horse" s2 = "ros"

1) Insert
2) Remove
3) Replace

Match string via 2-pointers

if (s1[i] == s2[j]) return 0 + f(i-1, j-1) // shrink.

{ 1 + f(i, j-1) → insert
  1 + f(i-1, j) → delete.
  1 + f(i-1, j-1) → replace. } min

Memoization:-

```
int f(i,j,s1,s2){
  (i==0) j
  (j==0) i
if(dp[i][j] != -1) return dp[i][j];
if(s[i-1] == t[j-1]) {
   return dp[i][j] = f(i-1,j-1,s,t)
}
return dp[i][j] = 1 + min( f(i-1,j,s,t), min f(i,j-1,s,t),
                           f(i-1,j-1,s,t)) ;
}
```

insert, del, replace

```
main()
dp(n+1, vector<int>(m+1,-1))
return f(n-1, m-1, s, t, dp)
```

Bottom-Up:-

```
n = s1.size(), m = s2.size()
for(i=0 <=n ) dp[i][0] = i;
for(j=0 <=m) dp[0][j] = j;

for(i=1; <=n ; i++)
    for(j=1; j<=m; j++)
        if(s[i-1] = s[j-1])
            dp[i][j] = dp[i-1][j-1]
        else dp[i][j] = 1+ min{ dp[i-1][j], dp[i][j-1]
                                dp[i-1][j-1]);

dp[n][m];
```

Wildcard Matching

s = "aa" p = "a"

return - false, we have to match two strings.

If p contains "*" → p can match with multiple char.

If p contains "?" → can match with single char.

s = "aa", p = "*" → O/P: True.

s = "cb", p = "?a" → false

? → c
b != a.

If (n-1, m-1) → Taking 2 pointers,

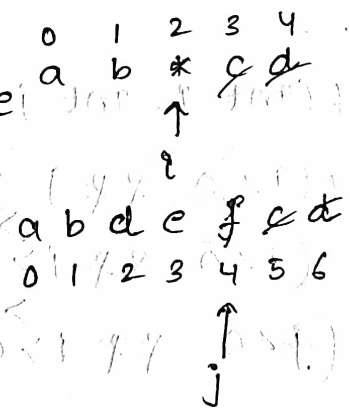
If (s1[i] == s2[j] || s1[j] == '?')

return f(i-1, j-1);

If p matches simply shrink the string.

Now, consider this case →

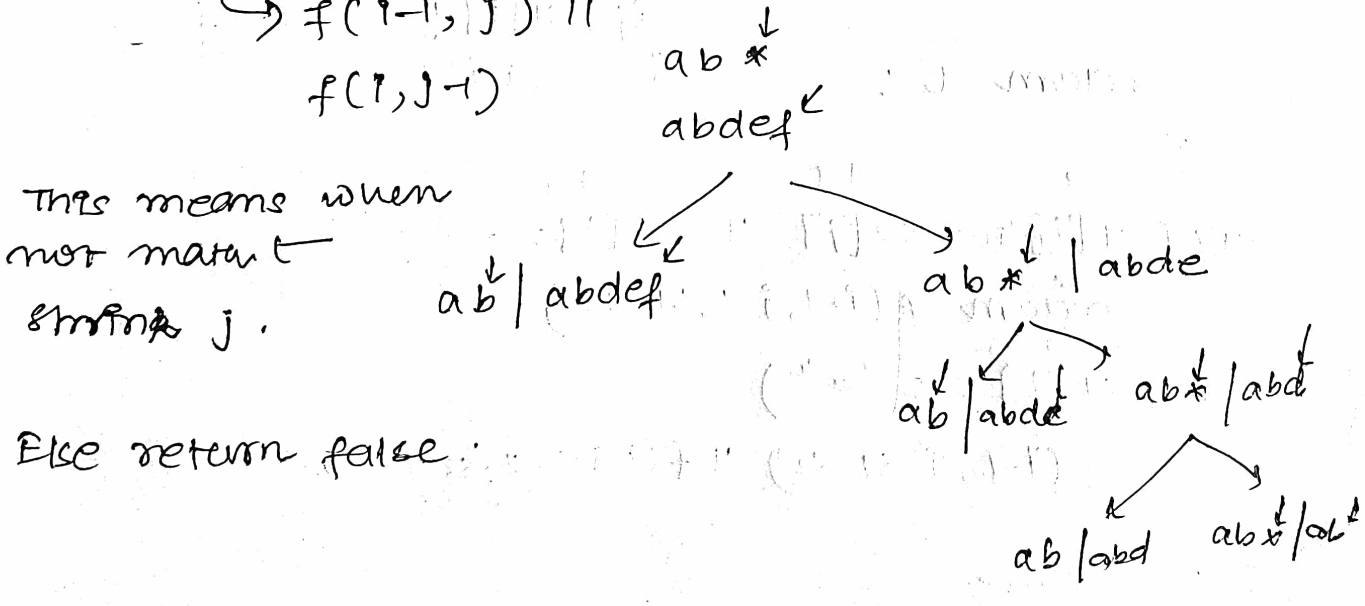
But now they aren't matching we would say "*" means nothing and shrink p (2 → 1)



If (s1[i] == '*')

↳ f(i-1, j) ||

f(i, j-1)



Else return false.

Base Case:- → if (T/F)

All comparison have been done.

if (i<0 && j<0) return true;

Both are exhausted

if (i<0 && j>=0) return false

i exhausted but j still remaining but no char remain to be matched.

if (j<0 && i>=0) for (0 → i)
if (s[i] != '*') return false;
return true;

If s1 is left it has to be all "*".

Time Complexity : O(exponential)

SC: O(N+M)

Code:-

```
int f(int i, int j, s1, s2)
  if (i<0 && j<0) true            → 1 based index
    (i<0 && j>=0) false
    (j<0 && i>=0){for (1 <= i) {
                 if (s1[i-1] != '*') → f
                 }
  return t;
}
if (s1[i-1] == s2[j-1] || s1[i-1] == '?')
  return f(i-1, j-1, s1, s2)
if (s1[i-1] == '*')
  f(i-1, j, s1, s2) || f(i, j-1, s1, s2)

return false
```

Bottom-Up

```
f ( s1, s2) {
  n = s1.size(), m = s2.size()
  vec < vec < bool > > dp(n+1, vec < bool > (m+1, false));
  dp[0][0] = true;
  for(j=1; j<=m; j++) dp[0][j] = false;
 for(i=1; i<=n; i++) dp {
      bool flag = true;
     for(j=1; j<=i; j++){
         if (s1[i-1] != '*')
             flag = false;
             break;
          }
      }
    dp[i][0] = flag;
}

 for( i=1 <=n) {
   for(j=1<=m) {
     if (s1[i-1] == s2[j-1] || s1[i-1] == '?') {
          dp[i][j] = dp[i-1][j-1]
     }
    else if ( s1[i-1] == '*') {
        dp[i][j] = dp[i-1][j] || dp[i][j-1]
   }
   else dp[i][j] = false;
  }
}
 return dp[n][m];
```