Longest Common Subseq. (LCS)

* Longest common Subsequence:-

1) Longest common substring
2) Print LCS
3) Shortest common super sequence
4) Print SCS
5) Min no of insertion & deletion to make a → b
6) Longest repeating subsequence.
7) Length of longest subsequence of a which is a substring in b.
8) Subsequence pattern matching.
9) Count how many times a appears as subsequence in b.
10) Longest palindromic subsequence.
11) Longest palindromic substring.
12) Count of palindromic substring.
13) Min no of deletion in a string to make it a palindrome.
14) Min no of insertion in a string to make it a palindrome.

* LCS Recursive-

I/P

X: a b c d g h

Y: a b e d f h r

O/P:- 4 (abdh)

n: length of X

m: length of Y

Flow

→ Problem sttmnt

→ Recursive solution

 ↳ Base condin

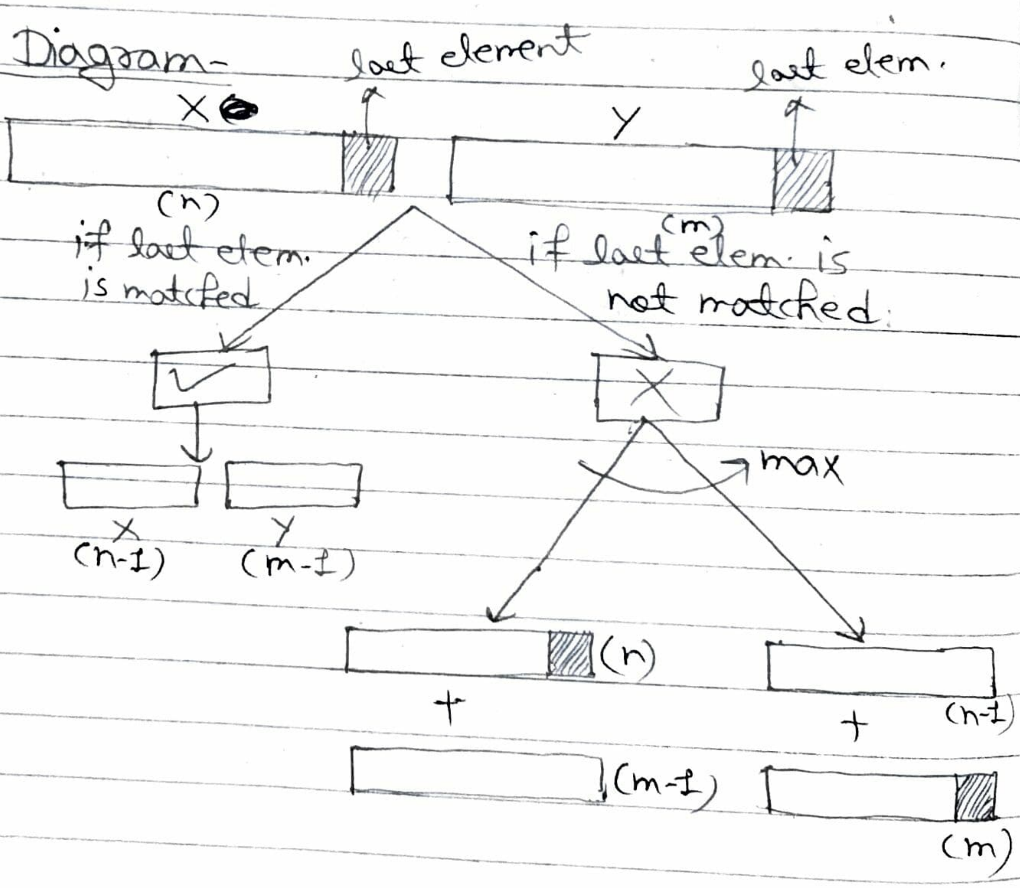
 ↳ choice diagram

→ code

Base condition-

```
if (n==0 || m==0)
    return 0;
```

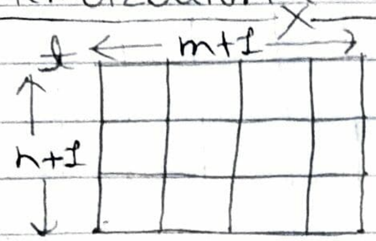
Choice Diagram-



Code-

```
    if (X[n-1] == Y[m-1])
        return (1+ LCS(X, Y, n-1, m-1));
    else
        return max(LCS(X, Y, n, m-1),
                          LCS(X, Y, n-1, m));
```

* Memoization (LCS) -



t[n+1] [m+1]

Base condition -

```
              if(t[n][m] != -1)
                  return t[n][m];
          if(n==0 || m==0) return 0;
```

Main code-

```
     ~~if(x[m-1]~~
     if(x[n-1] == Y[m-1])
         return t[n][m] = 1+ LCS(X, Y, n-1, m-1);
     else
     return t[n][m] = max( LCS(X, Y, n, m-1),
                               LCS(X, Y, n-1, m));
```

* LCS - Top Down

I/P

X: a b c d g h
Y: a b e d f h r

m: length of X = 6
n: length of Y = 7 → n

t[m+1][n+1]
t[7][8]

t	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							▨

↓ m

t[6][7]

```
for(int i=0; i<m+1; i++)
   for(int j=0; j<n+1; j++)
      if (i==0 || j==0)
         t[i][j] =0;

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

* 1- Longest Common Substring

I/P

a : abcde m: 5
b : abfce n: 5

common substrings are - "ab", "e", "c"
↓
longest

o/p = length of longest common substring

o/p = 2

Initialization

t

0	0	0	0
0			
0			
0			

t[m+1][n+1]

Code-

```
if (a[i-1] == b[j-1])
    t[i][j] = 1 + t[i-1][j-1];
else
    t[i][j] = 0;
```

* 2-Print LCS b/w 2 substrings -

I/P

a: acbcf	m:5
b: abcdaf	n:6

O/P:- abcf

n

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

m

Table after ~~memoization~~ Top Down

max

max

"fcba"

Code-

```
int i=m; j=n;
 String s = " ";
 while(i>0 && j>0)
 {
   if(a[i-1]==b[j-1])
   { s.pushback(a[i-1]);
     i--;
     j--;
   }
 else
   { if (t[i][j-1] > t[i-1][j])
        j--;
     else
        i--;
   }
 }       printf(reverse(s));
```

sequence → order maintained but it is not important that it is continuous.

* 3- Shortest Common SuperSequence -

~~I/P~~

a: "geek"
b: "eke"

Given two string str1 & str2, the task is to find the length of the shortest string that has both str1 & str2 as subsequence.

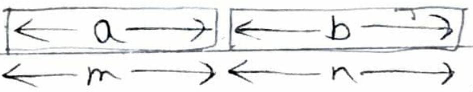
Ex- a: "geek"
b: "eke"

we can merge two string in many ways -
geeke , geekeke

Ex- a: "AGGTAB"
b: "GXTXAYB"

By merge -
AGGTGX ABTXAYB , AGGXTXAYB ✓

Approach -



= m+n

$= (m+n) - LCS$

GTAB   G~~T~~AB

$\underbrace{\text{GTAB}}_{\text{LCS}}$

= SCSS ✓

*4-Print Shortest common supersequence-

IP: a: acbcf
b: abcdaf

OP: "acbcdaf"

It is similiar to print LCS!

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

→ Table of LCS

if we stop here

a: acbcf
b: abcdaf] : LCS = abcf

→ then we have a: ac
b: φ (or) " "

LCS=0   SCSS=ac

Code-

```
int i =m, j=n;
String s=" ";
while(i>0 && j>0)
{
    if(a[i-1] == b[j-1])
    {  s.pushback(a[i-1]);
       i--;
       j--; }
else {
    if (t[i][j-1] > t[i-1][j]) {
        ~~j--;~~
                    s.pushback(b[j-1]);
    ~~else~~
        ~~i--;~~    j--; }
    ~~}~~

     else if (t[i-1][j] > t[i][j-1])
      {  s.pushback(a[i-1]);
         i--;
      }
    }
}
while (i>0){
      s.pushback(a[i-1]);
      i--;
  }
while (j>0) {
      s.pushback(b[j-1]);
      j--;
  }
```

* 5- Minimum no of Insertion & Deletion to convert string 'a' to string b

I/P

a: heap
b: pea

O/P-

Insert - 1
Delete - 2

a → b

(b) heap → pea

LCS

heap → pea

2 deletion

1 insertion

LCS(ea)

len(a) - LCS

len(b) - LCS

result = (len(a) - LCS) + (len(b) - LCS)

= (4 - 2) + (3 - 2)

= 2 + 1

= 3

* 10- Longeet Palindromic Subsequence - (LPS)

I/P $\boxed{S: agbcba}$ O/P $\boxed{5}$

~~Subsets~~ Subsequence
↓
Palindromic
↓
longeet

This queetion is based on LCS, but we have only one etoing 'S'. To solve using LCS we muet have 2 etoings! So we derive 2nd etong using 1st etoing.

Let,
a : agbcba
b : abcbga → reverse of a

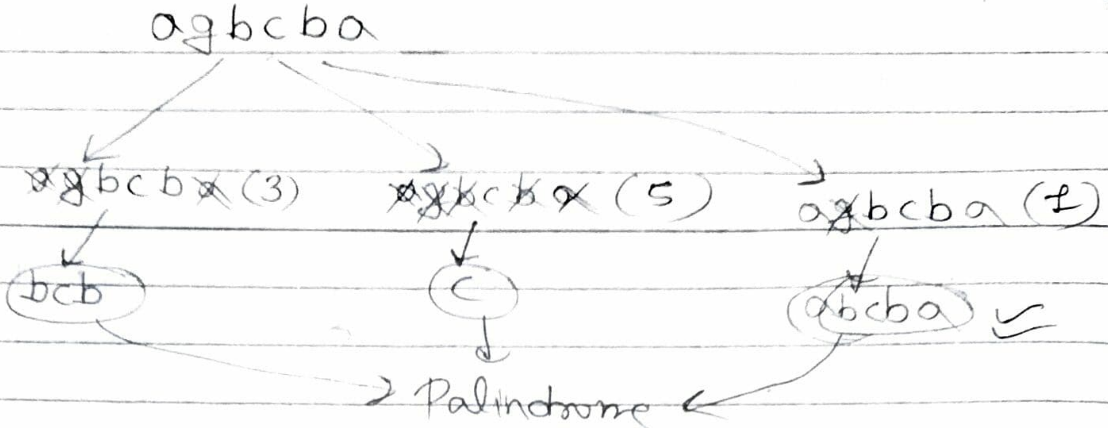
Now we can apply LCS -

LCS of a & b ⇒ "abcba" which is also LPA

⇒ $\boxed{LPS(a) \equiv LCS(a, reverse(a))}$

* 13- Min no of deletion in a string to make it a palindrome

I/P: $\boxed{s: \text{"agbcba"}}$ O/P: $\boxed{1}$



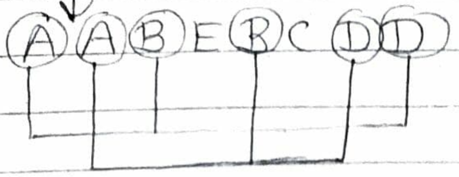
$$\boxed{\uparrow \text{Length of LPS} \propto \frac{1}{\text{No of deletion} \downarrow}}$$

$$\boxed{\text{Min no of deletion} = \text{length of string} - \text{LPS}}$$

* 6- Longest repeating subsequence-

str = "AABEBCDD"   O/P :- 3

subsequence → order, → Discontinuous



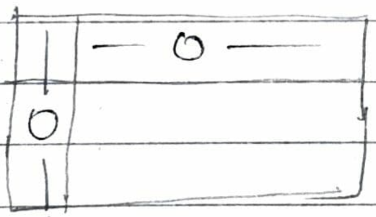
→ ABD → 2X times

subsequence occurs 2 times

Approach-

a: AABEBCDD
b: AABEBCDD
LCS (i != j)

Initialization-



Code-

```
if( a[i-1] == b[j-1] && i != j )
    t[i][j] = t[i-1][j-1] + 1
else
    t[i][j] = max( t[i][j-1],
                   t[i-1][j] )
```

* 8- Sequence Pattern Matching

a : "AXY"
b : "ADXCPY"

O/P :- T/F : T

→ is "a" a subsequence of "b"

Approach a : AXY
b : ADXCPY] → LCS == a/b

```
Find LCS then,
  if (LCS == a.length())
      return True
  else
      return False
```

* 14- Min No. of insertion in a string to make it a palindrome-

I/P

S: "aebcbda" O/P : 2

a e b c b d a

↙ ↘

adebcbeda xadebcbedax

(2) (4)

Approach- Let s : $\boxed{a}\,e\,\boxed{b\,c\,b}\,d\,\boxed{a}$

To make s palindrome, we can delete some elements or we can insert some elements.

Previously we already studied what is 'min no of deletion to make string palindrome'

If we insert those elements which we want to delete to make palindrome, then we can make string palindrome.

⇒ No of insertion = No of deletion