Agenda

1) longest common subsequence

2) longest palindromic subsequence

3) Edit distance

Q.1 Given 2 strings, find length of longest common subsequence.

eg.1

	0	1	2	3	4	5	6
S1 =	a	b	b	c	d	g	h

	0	1	2	3	4	5	6
S2 =	b	a	c	h	e	g	j

ans = 3 (acg or bcg)

eg.2

	0	1	2	3	4	5	6
S1 =	a	b	b	c	d	g	j

	0	1	2	3	4	5
S2 =	a	c	h	e	g	j

ans = 4 (acgj)

Brute force : find all subsequence of S1 and fill them in an ArrayList, do the same thing for S2. find longest common sequence now.

S1 = a b c

S2 = a c e

u1 → ·, a, b, c, ab, ac, bc, abc

u2 → ·, a, c, e, ac, ae, ce, ace

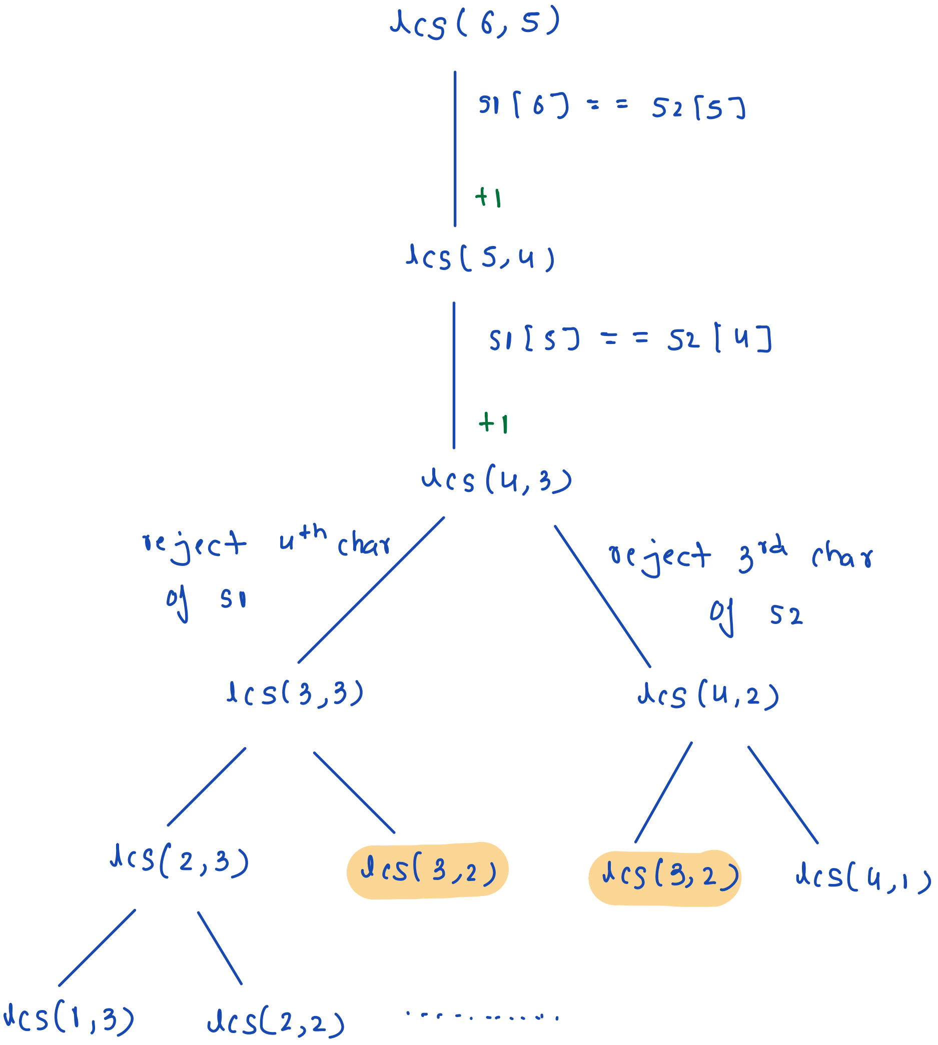
TC: $O(2^n)$

common subseq → ·, a, c, ac

Improvise

	0	1	2	3	4	5	6
s1 =	a	b	b	c	d	g	f

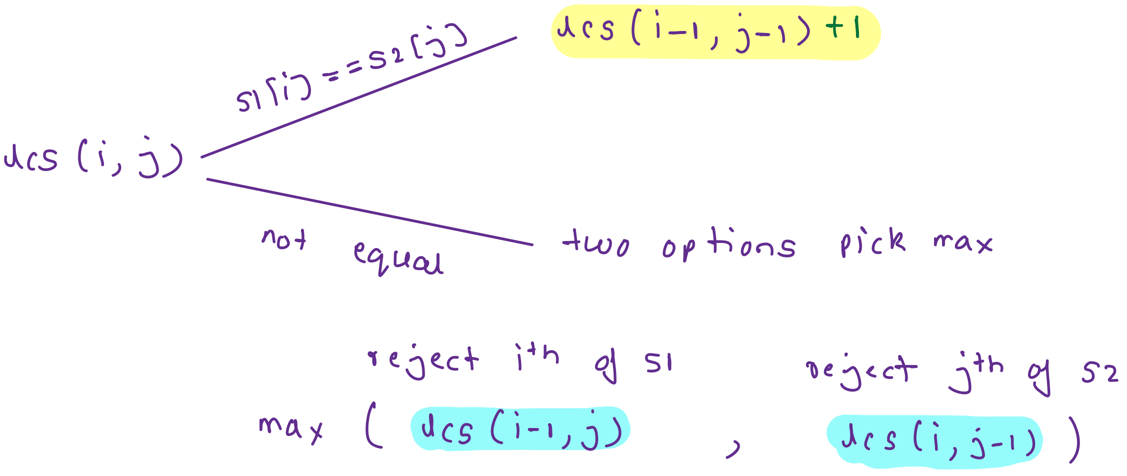
	0	1	2	3	4	5
s2 =	a	c	h	e	g	f



dp can be applied

dp[][] → dp[n1][n2]

```
        0 1 2 3 4 5 6
S1 =    a b b c d g j

        0 1 2 3 4 5
S2 =    a c h e g j
```



lcs (i, j)

s1[i] == s2[j]

lcs (i-1, j-1) +1

not equal

two options pick max

reject ith of s1

reject jth of s2

max (lcs (i-1, j) , lcs (i, j-1))

```
int solve ( string s1, string s2) {
      int n1 = s1. length();
       int n2 = s2. length();
        dp= new int [n1] [n2];
        // fill dp with -1
        return   lcs ( s1, s2, n1-1, n2-1);
}
```

```
int [ ] [ ] dp;

int   lcs ( string s1, string s2, int i, int j) {
    if ( i<0 || j<0) {
          return 0;
    }

    if( dp [i] [j] != -1) {
          return dp [i] [j];
    }

    int  ans = 0;

    if (s1.charAt (i) == s2.charAt(j) ) {
          ans= lcs ( s1, s2, i-1, j-1) +1;
    }
    else {
          int  a= lcs (s1, s2, i-1, j));
          int  b= lcs (s1, s2, i, j-1) ;
          ans= Math.max (a,b);
    }
    dp [i] [j] = ans;
    return  ans;
}
```

TC : O(n1 * n2)

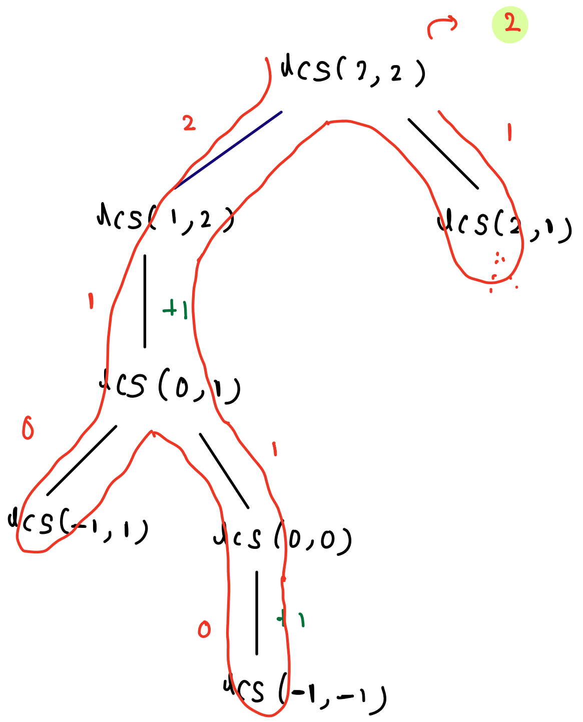
SC : O(n1 * n2)

dry run → to understand recursion

```
int   lcs ( string s1, string s2, int i, int j) {
   if ( i<0 || j<0) {
         return 0;
   }

   int ans= 0;

   if (s1.charAt(i) == s2.charAt(j)) {

         ans= lcs ( s1, s2, i-1, j-1) +1;

   }
   else {

         int a= lcs (s1, s2, i-1, j);

         int b= lcs (s1, s2, i, j-1);

         ans= Math.max (a,b);

   }

   return ans;
}
```

$s1 = \overset{0}{a}\ \overset{1}{d}\ \overset{2}{c}$

$s2 = \overset{0}{a}\ \overset{1}{e}\ \overset{2}{d}$



Tabulation of lcs :

dp= new int [n1][n2];

→ **Relation b/w problem & subproblem**

```
int ans= 0;

if (s1.charAt(i) == s2.charAt(j)) {

        ans= lcs ( s1, s2, i-1, j-1) +1;

}
else {

      int a= lcs (s1, s2, i-1, j));
      int b= lcs (s1, s2, i, j-1) ;

      ans= Math.max(a,b);

}
```

	0	1	2	3	4	5	6
s1 =	a	b	b	c	d	g	f

	0	1	2	3	4	5
s2 =	a	c	h	e	g	f

		a	c	h	e	g	f
		0	1	2	3	4	5
a	0	1	1	1	1	1	1
b	1	1	1	1	1	1	1
b	2	1	1	1	1	1	1
c	3	1	2	2	2	2	2
d	4	1	2	2	2	2	2
g	5	1	2	2	2	3	3
f	6	1	2	2	2	3	4

dp[i][j] = lcs of (i, j)

lcs of <u>s1 till i</u> and <u>s2 till j</u>

TC: O(n1 * n2)

SC: O(n1 * n2)

Q.2 Given a String, find longest palindromic subsequence.

A = scalar   ans = 3

A = a b d c e f b   ans = 3

A = a d b c g b a   ans = 5

A = a d b c g b a

A' = a b g c b d a   ans = 5

reverse of A

longest palindromic subsequence is nothing but lcs (string, reverse-of-string)

A = scalar

A' = ralacs

ans = 3

A = a b d c e f b

A' = b f e c d b a

ans = 3

Q.3 Edit distance {G favourite}

Given 2 strings s1 and s2, min operations to be performed in s1 so that it becomes equals to s2.

operations allowed in s1 :

i) we can **insert** any char in s1 at any position

ii) we can **replace** any char in s2 at any position

iii) we can **delete** any char in s2 at any position

eg.1

s1 = ~~d~~ f $\overset{g}{\cancel{d}}$ ~~c~~ d

s2 = f g d

ans = 3

eg.2

s1 = ~~d~~ f ~~a~~ ~~e~~ x z

s2 = f x z

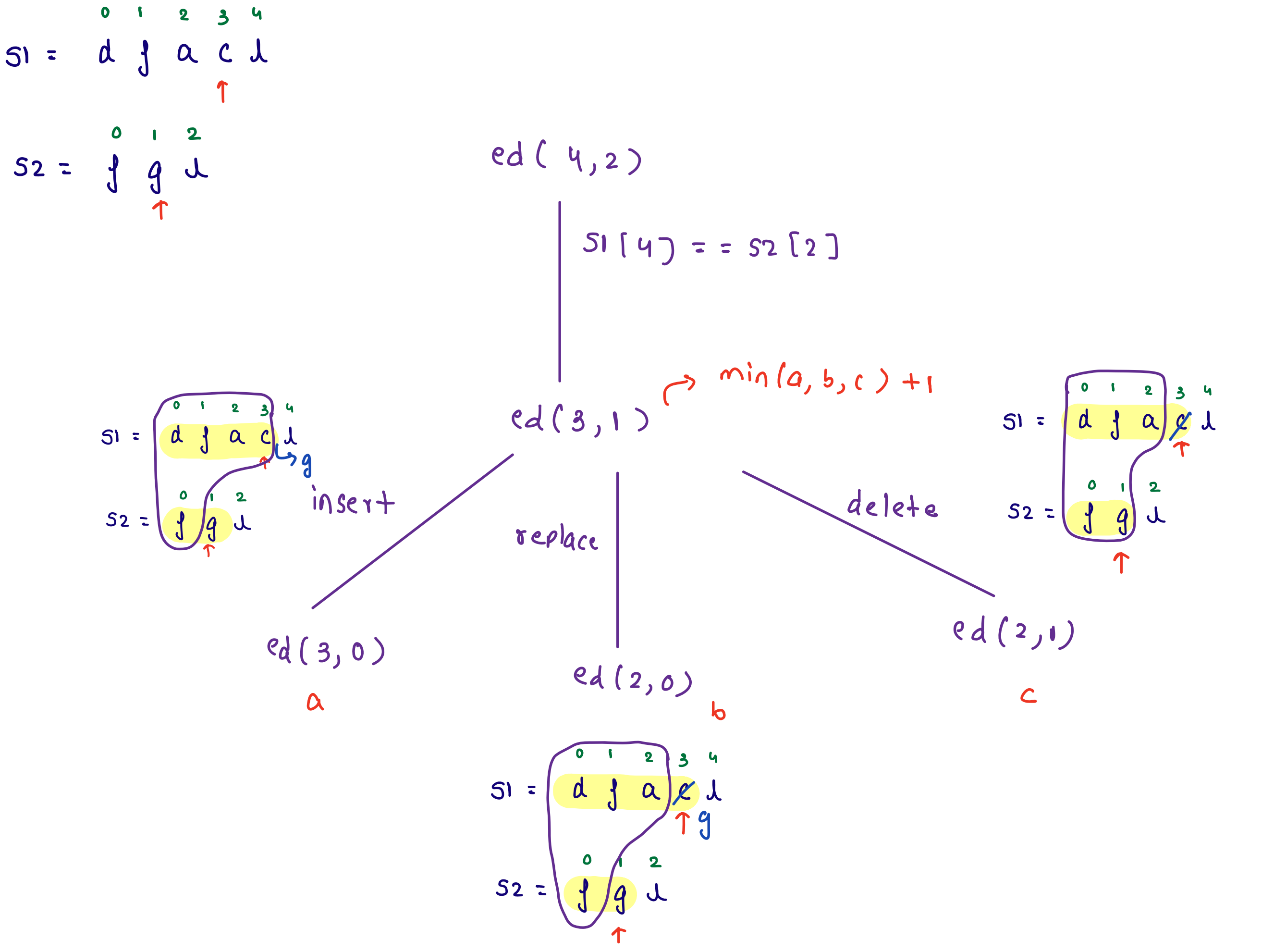
ans = 4

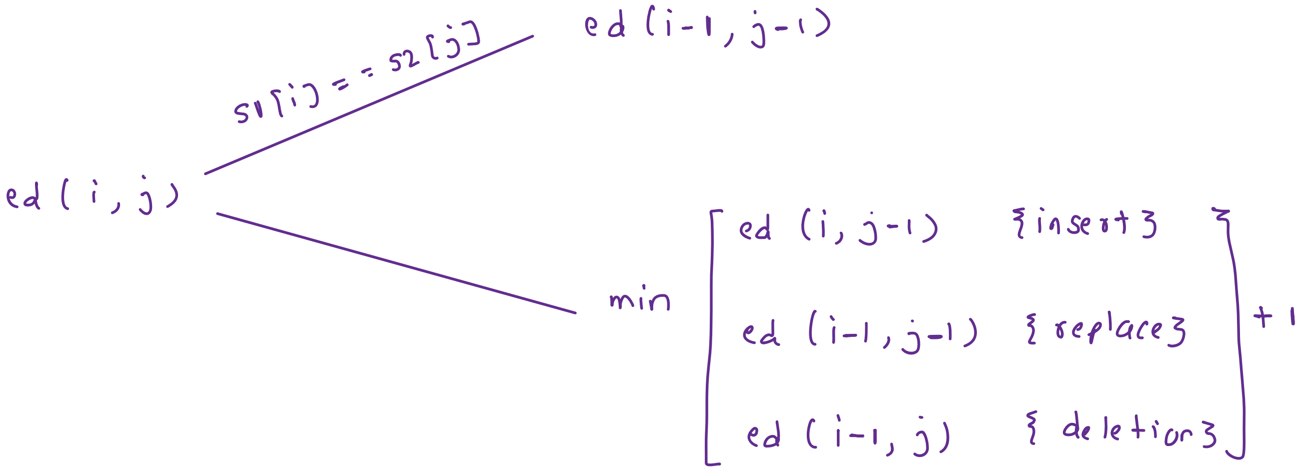
ans = s1.length() - lcs of s1, s2 this logic won't work

(why : check out 2nd example)

s1 = d $\underline{f}$ a e $\underline{x}$ lcs = 2   ans = 5-2 = 3 ✗

s2 = $\underline{f x}$ z


0 1 2 3 4
S1 = d j a c d
0 1 2
S2 = j g d
ed (4,2)
S1 [4] = = S2 [2]
ed (3 , 1)
min (a, b, c) +1
insert
replace
delete
g
ed (3, 0)
a
ed (2, 0)
b
ed (2, 1)
c


ed (i , j)
S1 [i] = = S2 [j]
ed (i-1 , j-1)
min
ed (i, j-1) { insert }
ed (i-1, j-1) { replace }
ed (i-1, j) { deletion }
+ 1

Memoization (DP applied in recursive code)

```
int solve (string s1, string s2) {
    int n1 = s1.length();
    int n2 = s2.length();
    dp= new int [n1][n2];
    // fill dp with -1
    return ed(s1, s2, n1-1, n2-1);
}

int [][]dp;
int  ed (string s1, string s2, int i, int j) {

   if (i<0) → return j+1;      { j+1 insertions needed in s1 }

   if (j<0) → return i+1;      { i+1 deletions needed in s1 }


   if (dp[i][j]!=-1) {
        return dp[i][j];
   }
   int ans=0;
   if (s1.charAt(i) == s2.charAt(j)) {
         ans= ed (s1, s2, i-1, j-1);
   }
   else {
         int a= ed(s1, s2, i, j-1); // insert
         int b= ed (s1, s2, i-1, j-1); // replace
         int c= ed (s1, s2, i-1, j); //deletion
         ans= Math.min (a, Math.min (b,c)) +1;
   }
   dp[i][j] = ans;
   return ans;
}
```

TC: O(n1*n2)

SC: O(n1*n2)

Tabulation

$$ed(i,j) \begin{cases} \xrightarrow{s1[i] == s2[j]} ed(i-1,j-1) \\ \min \left[\begin{array}{ll} ed(i,j-1) & \{insert\} \\ ed(i-1,j-1) & \{replace\} \\ ed(i-1,j) & \{deletion\} \end{array}\right] + 1 \end{cases}$$

$$s1 = \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ d & f & a & e & x \end{array}$$

$$s2 = \begin{array}{ccc} 0 & 1 & 2 \\ f & x & z \end{array}$$

	f 0	x 1	z 2
d 0	1	2	3
f 1	1	2	3
a 2	2	2	3
e 3	3	3	3
x 4	4	3	4

code : todo

dp[i][j] => ans of (s1 till i, s2 till j)