Today's content (Strings Based DP)

- Longest common subsequence
- Edit distance
- Regex matching

Q. Given 2 strings, Find length of longest common subsequence (LCS (S_1, S_2))

	0	1	2	3	4	5	6
S_1	a	b	b	c	d	g	f
S_2	b	a	c	h	e	g	f

ans = 4

Seq 1: a c g f

Seq 2: b c g f

S_1	k	l	a	g	r	i
S_2	l	g	i	g	k	m

lgi

ans = 3

Lcs (S_1[0-6], S_2[0-6])

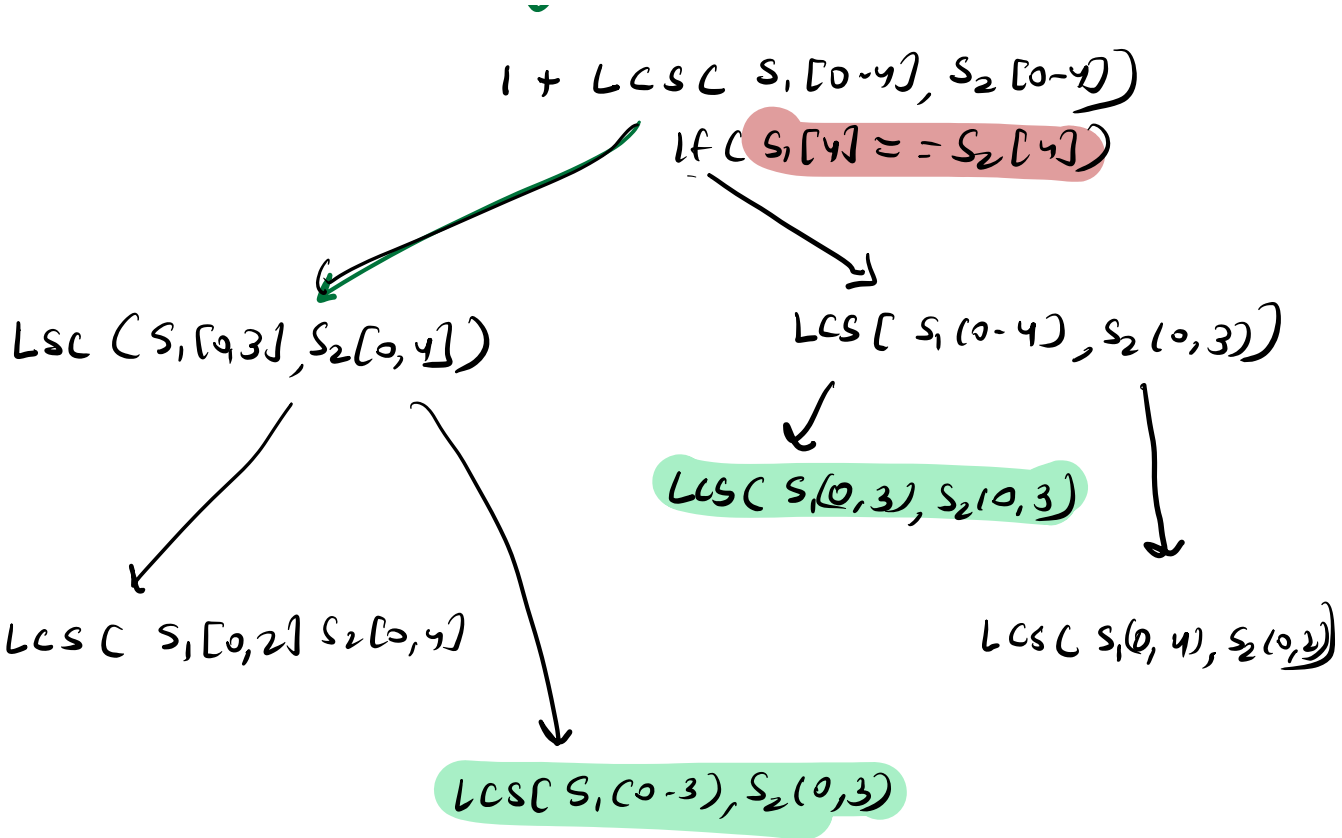
→ optimal substructure

→ overlapping subproblem

If (S_1[6] == S_2[6])

If LCS (S_1[0-5], S_2[0-5])

If (S_1[5] == S_2[5])



dp state

i (starting at 0)
representing ending index of S_1

dp[i][j]

j representing ending index of S_2

len of LCS of $S_1[0-i]$ & $S_2[0-j]$

$$dp[i][j] = \begin{cases} \text{if } (S_1[i] == S_2[j]) \\ \quad 1 + dp[i-1][j-1] \end{cases}$$

else

$\max(dp[i-1][j], dp[i][j-1])$

Recursive code

```
int dp[N][M] = {-1}
return LCS(S1, S2, N-1, M-1, dp)

int LCS(S1, S2, i, j, dp[][])
{
    if(i == -1 || j == -1) return 0

    if(dp[i][j] == -1)
    {
        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]
}
```

	0	1	2	3
a				
a	b	c	d	

dp[0][3]

dp[-1][3]

i = -1

j != -1

i = -1 j = -1

Tc : O(n x m x 1) ≈ O(N x m)

Sc : O(NM)

Tracing

		0	1	2	3	4
			M	A	I	C	A
			I	A	I	Y	A	S

$$dp[i][j] = \begin{cases} \text{if } (s_1[i] == s_2[j]) \\ \{ \quad 1 + dp[i-1][j-1] \\ \text{else} \\ \{ \quad \max(dp[i-1][j], \; dp[i][j-1]) \end{cases}$$

dp[5][6]

		0	1	2	3	4	5
		I	A	I	Y	A	S
0	M	0	0	0	0	0	0
1	A	0	1	1	1	1	1
2	I	1	1	2	2	2	2
3	C	1	1	2	2	2	2
4	A	1	2	2	2	3	3

A I A

```
string ans = ""
int i = n-1, j = m-1

while( i >= 0 && j >= 0)
{
    if( s1[i] == s2[j])
    {
        // s1[i] is present in ans
        ans += s1[i]
        i--, j--
    }
    else
    {
        if(i>0 && dp[i][j] == dp[i-1][j]) i--
        else j--
    }
}
return rev(ans)
```

Q. Edit distance.

Given 2 string s_1, s_2. min operations to be performed on s_1 so that s_1 becomes s_2

In 1 operation of s_1

- we can insert a character in s_1 at any point
- we can replace any char with any char in s_1
- we can delete any char in s_1

ans = 3

$S_1 =$ d f a e f

$S_2 =$ f g f

S_1 ~~x~~ f a ~~x~~ f

~~x~~ f ~~x~~ e f

S_1 : f e h

S_2 : a e k f

a f e k h f

ans = 3

$S_1 =$ d f a e f

$S_2 =$ f g f

Ed(S_1[0-4] , S_2[0-2])

↓ if (S_1[4] == S_2[2])

Ed(S_1[0-3], S_2[0-1])

if (S_1[3] != S_2[1])

Ins of S_1 ↙ | Replace, ↓ | Delete S_1 ↘

Ed(S_1[0-3], S_2(0,0)) + 1

1 + Ed(S_1(0-2) S_2(0-0))

1 + Ed(S_1(0-2) S_2(0-1))

dfaeg

fg

0 1 2 3 ... i-1 i

j

0 1 2 ... j-1 j

$dp[i][j]$ { min operation on $S_1[0-i]$ to make it $S_2[0-j]$

$$dp[i][j] = \begin{cases} \text{if } (S_1[i] == S_2[j]) \\ \quad dp[i-1][j-1] \\ \text{else} \\ \quad 1 + \min \begin{cases} dp[i][j-1] & \text{// insert} \\ dp[i-1][j-1] & \text{// replace} \\ dp[i-1][j] & \text{// delete} \end{cases} \end{cases}$$

Pseudo code

$dp[N][M] = \{-1\}$

```
int Ed(S1, S2, i, j, dp)
{
    if (i == -1 && j == -1)   return 0
    if (i == -1)
    {  return j+1 }   { j+1 insertions }
    if (j == -1)
    {  return i+1 }   { i+1 deletions }

    if (dp[i][j] == -1)
    {
        if (S1[i] == S2[j])
            dp[i][j] = EDC(S1, S2, i-1, j-1, dp)
        else
        {
            dp[i][j] = 1 + min( EDC(S1, S2, i-1, j-1, dp),
                                EDC(S1, S2, i  , j-1, dp),
                                EDC(S1, S2, i-1, j  , dp) )
        }
    }

    return dp[i][j]
}
```

TC : $O(N \times M)$

SC : $O(N \times M)$

Q. Regex matching

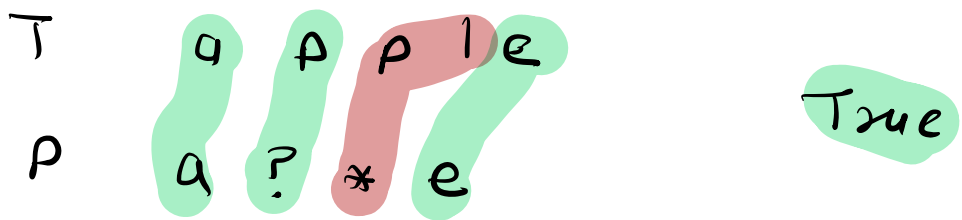
Given Text (T) & Pattern (P), check

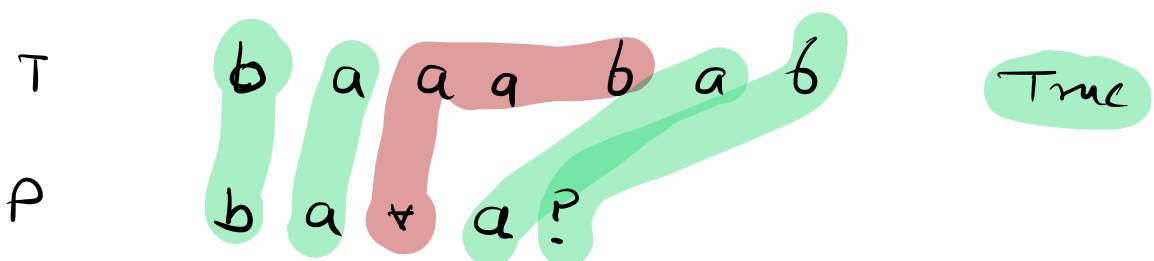
if both are same or not

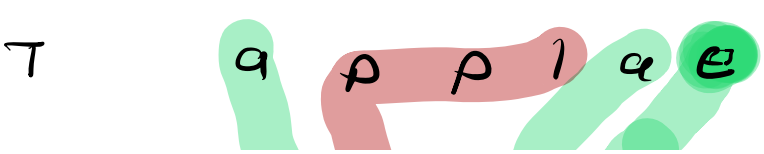
T → only alphabets.
P → alphabets, ?, *

? → Can match any 1 char
* → Can match any no. of char

Note: any no of ? * can be there ≥ 0







p a * a ? e — False

a p p l e

a * a ? e

T a n t

p a ? * * * t — True

T " "

p * * * — True

T " "

p ? — False

T : 0 1 2 3 4
a p p l e

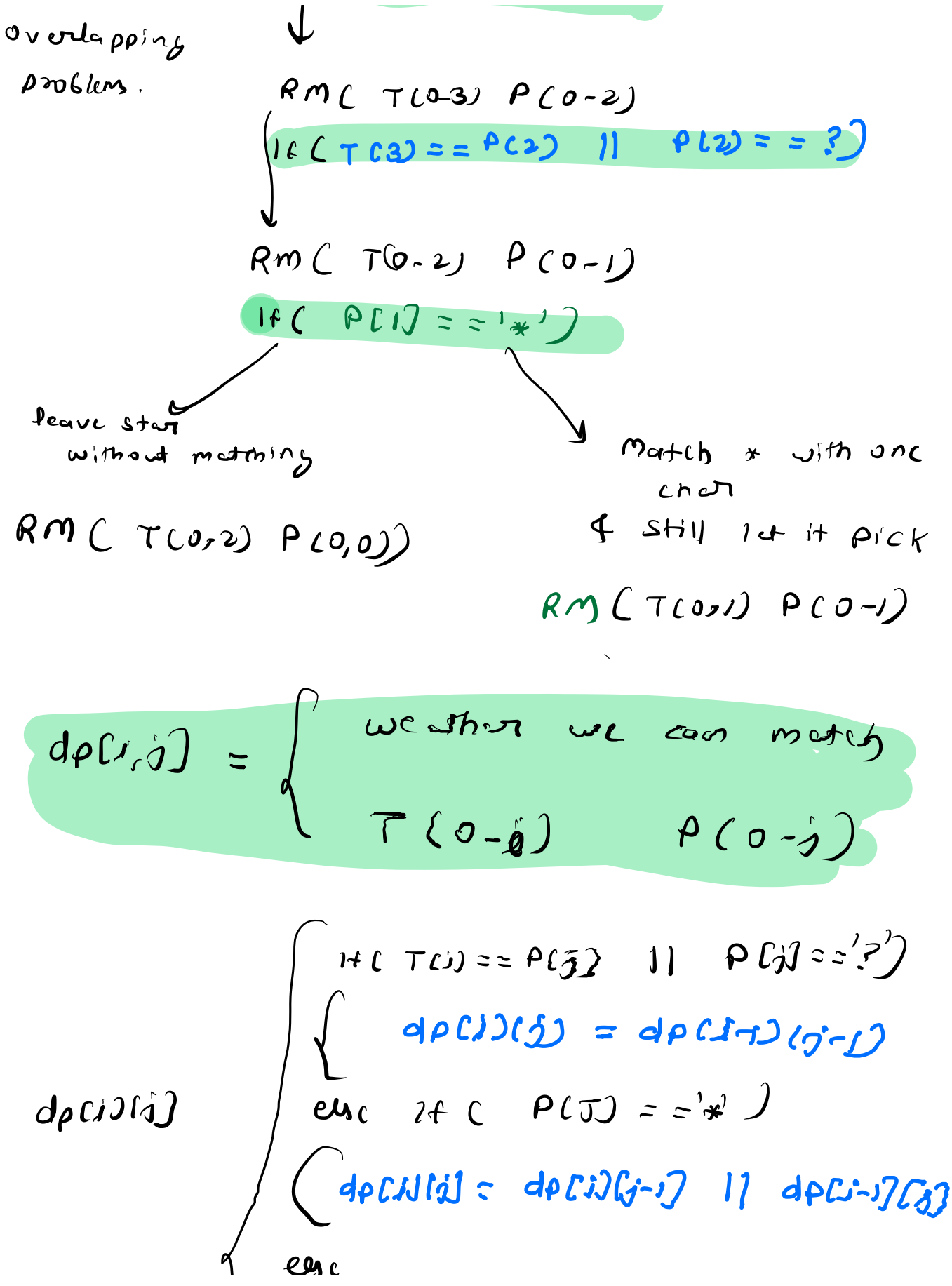
P : a * ? e

optimal substructure

RM (T(0-4) P(0-3)

| if(T(4) == P(3)

overlapping problems.



$$dp[i,j] = \begin{cases} \text{whether we can match} \\ T(0\text{-}i) \quad P(0\text{-}j) \end{cases}$$

$dp[i][j]$

- if($T[i] == P[j]$ || $P[j] == '?'$)
  - $dp[i][j] = dp[i-1][j-1]$
- else if($P[j] == '*'$)
  - $dp[i][j] = dp[i][j-1]$ || $dp[i-1][j]$
- else

```
{   dp[i][j] = false.
```

```
int  dp[N][M] = -1
return  RM(T, P, N-1, M-1, dp)
```

```
int  RM( T, P, i, j, dp)
{
  if (i==-1 && j==-1)   return 1
  if (j==-1)   return 0

  if (i==-1)
  {
     for(k=0 ; k<=j ; k++)
     {   if (P[k] != '*')   return 0
     return 1 ;

  if (dp[i][j] == -1)
  {
     if (T[i]==P[j] || P[j]=='?')
          dp[i][j] = RM(T,P,i-1,j-1,dp)
     else if(P[j]=='*')
      dp[i][j] = RM(T,P,i,j-1,dp) || RM(T,P,i-1,j,dp)
     else
          dp[i][j] = 0
```

TC : O(N*M)

SC : O(N*M)