Today's Agenda : Starting 7:05

1. LCS of two strings.
2. Edit distance
3. String Pattern Matching

Longest Common Subsequence (LCS)

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

acdgf
OR
bcdgf.

5

N S1 : d e m o c r a t
m S2 : r e p u b l i c a n

eca

3

BF idea :- Consider all subsequences of string S1. Store them. Consider all S.S. of S2 & find the longest one.

T.C. $O(2^N + 2^m * N)$.

N N-1

S1 _ _ _ _ a

S2 _ _ _ _ b.

m m-1

a b c d' e'

a c d' e'.

LCS (S1, (0, N-1) , S2 (0, m-1))

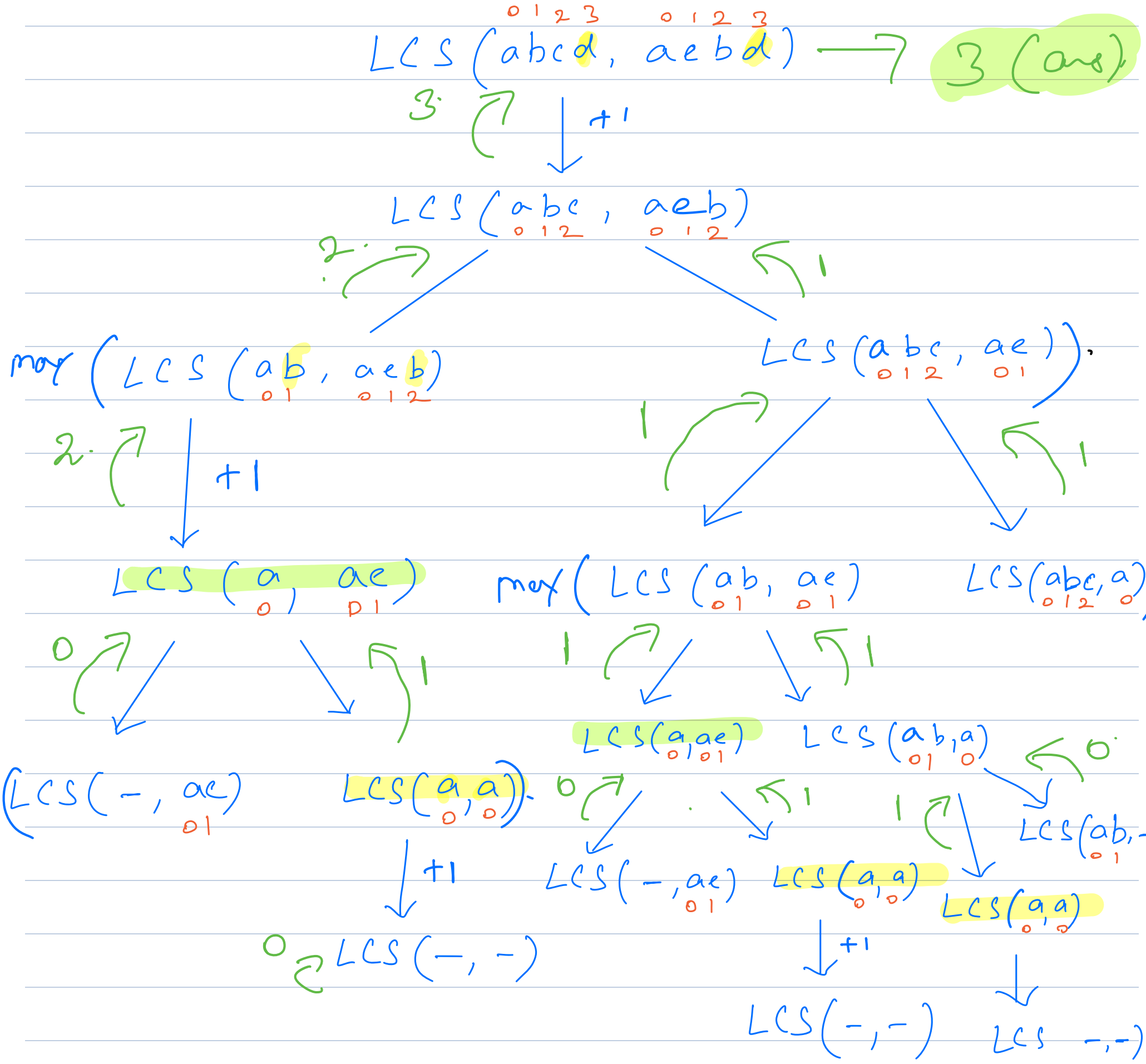
S1[N-1] == S2[M-1].

1 + LCS (S1 (0, N-2), S2(0, m-2)).

S1[N-1] != S2[M-1]

max (LCS(S1 (0, N-2), S2 (m-1)),

LCS (S1 (0, N-1) , S2(0, m-2)))



dp[i][j] → L.C.S of (0, i) of S1 & (0, j) for S2.

dp[N][M].

∀ i, j dp[i][j] = -1;

Code

N-1 ↓ , m-1 ↗

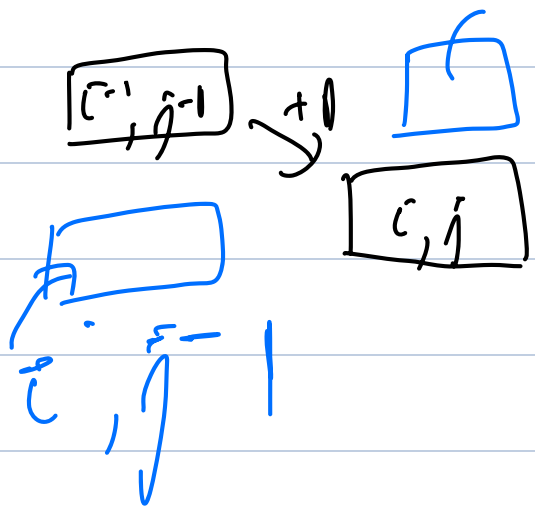
```
int lcs(s1, s2, i, j, dp[N][M]) {
  if (i<0 || j<0) return 0;
  if (dp[i][j] != -1) {return dp[i][j]};
    if (s1[i] == s2[j]) {
      dp[i][j] = lcs(s1, s2, i-1, j-1) + 1
    } else {
      dp[i][j] = max(lcs(s1, s2, i-1, j)
                   , lcs(s1, s2, i, j-1))
    }
  return dp[i][j];
}
```

T.C → O(N*m)
S.C. → O(N*m)

Botton Up approach:-

dp[i][j] → LCS of first i character of S1 & first j character of S2.

dp[0][0] → 0.



S1 = a e b d

S2 = a b c d

		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

s1[i-1] == s2[j-1] ; dp[i][j] = dp[i-1][j-1] + 1;

s[i-1] ≠ s2[j-1] ; dp[i][j] = max (dp[i][j-1], dp[i-1][j])

Code:

dp[N+1][m+1]

Initialise 0th row & 0th col 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).

↳ O(m). → H.W.

ms Google.

Edit Distance

Given S_1 & S_2. Convert $S_1 \rightarrow S_2$ by using some operations in S_1 only.

1. insert $\rightarrow c_i$
2. delete $\rightarrow c_d$
3. Replace $\rightarrow c_r$

Min Cost to convert S_1 to S_2.

$c_i = 2$; $c_d = 2$

$c_r = 3$.

Eg 1. $S_1 \rightarrow ac$ (b)
$S_2 \rightarrow abc$

2.

$2+2+2 = 6$.

Eg 1. $S_1 \rightarrow abcd$ (e)
$S_2 \rightarrow abe$

$2+3 = 5$

c
↓ g.

Eg 1. $S_1 \rightarrow abdxy$.
$S_2 \rightarrow abcgx$

2D + 2I
$= 6$.

1D + 1R + 1I
$= 7$.

S1 - - - -- a

S2 - - - - b

minCost(S1(0, N-1), S2(0, m-1))

S1[N-1] == S2[m-1]

S1[N-1] != S2[m-1]

minCost(S1(0, N-2), S2(0, m-2))

Minimum

Insert

Replace

Delete

Ci + minCost(S1, (0, N-1), S2, (0, m-2))

Cr + minCost(S1, (0, N-2), S2, (0, m-2))

Cd + minCost(S1, (0, N-2), S2, (0, m-1))

dp[N][m];

if dp[i][j] != -1;

Code - Top Down

int minCost(s1, s2, i, j, dp[N][M]) {

if (i<0 && j<0) {return 0;}

else if (i<0) {return (ci * (j+1));}

else if (j<0) {return (cd * (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,);

}

else {

$$dp[i][j] = \min \begin{cases} c_i + minCost(s1, s2, i, j-1, \\ c_d + minCost(s1, s2, i-1, j, \\ c_r + minCost(s1, s2, i-1, j-1, \end{cases}$$

return dp[i][j];

}

S_1	—	—	abcd.
S_2	—	abcd.	—
	0.	$c_i * (j+1)$	$c_d * (i+1)$

Bottom Up ⇒ H.W.

S2.

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

S1

dp[i][j]

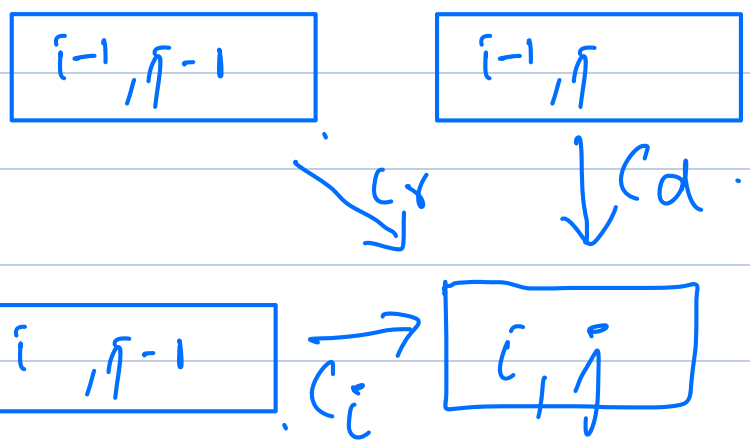
↓

minCost to convert first i characters of string S1 to first j characters of string S2.

dp[N][M] → ans.

if (S1[i-1] == S2[j-1])
{ dp[i][j] = dp[i-1][j-1]};

else:



dp[i][j] = min{ Cr + dp[i-1][j-1],
Ci + dp[i][j-1],
Cd + dp[i-1][j] 3;

9:02.

Wildcard Pattern Matching

Given S_1 & S_2. Check if they are matching.

S_2 → it can contain '?', '*'

'?' matches a single character

'*' matches with 0 or more characters

① S_1 → a b a c d
S_2 → a b a c d
true.

② S_1 → a b a c d
S_2 → a ? a c ?
true.

③ S_1 → x b b z c
S_2 → * * z *
true.

④ S_1 → x b b z z
S_2 → x * z * * * ? z
false.

⑤ S_1 → x b b z z
S_2 → x * z * * *
true.

check (S1 (0, N-1) , S2 (0, M-1))

S1[N-1] == S2[M-1].
OR
S2[M-1] == '*'

S1[N-1] != S2[M-1]

check (S1 (0, N-2), S2 (0, M-2))

return false;

S2[M-1] == '*'

match * with 0 character.

Match with 1 character & further

check (S1 (0, N-1) , S2 (0, M-2)).

check (S1 (0, N-2) , S2 (0, M-1)).

↑ t.

check (abc, *)

f. ↑

t. ↰

(abc, -)

(ab, *).

f. ↑

↰ t.

(ab, -)

(a, *)

f. ↑

↰ t.

(a, -)

(-, *)

dp[i][j] → check if s1(0,i) == s2(0,j)

dp[N][m] → ans:-

∀ dp[i][j] = -1

N-1 m-1

```
int check (s1, s2, i, j, dp[N][m]) {
if (i<0 && j<0) {return 1;}
else if (i<0 && CheckAllStars(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 );
    }
else if (s2[j] == '*') {
    dp[i][j]= max{ check(s1, s2, i, j-1 )
                 { check(s1, s2, i-1, j ),
3.
else
      dp[i][j] = 0;
    return dp[i][j];
3
```

T.C → O(N*m)

S.C → O(N*m)

```
checkAllStars(String s, int index) {
    int r = 0;
    for(i = 0; i <= index ; i++) {
        if(s[i] != '*')
            return false;
    }
    return true;
}
```

s1	—	—	$c_1c_2c_3c_4$	c_1c_2
s2.	—	$c_1c_2c_3c_4$.	—	c_3c_4.
	1	0 (check for all asterisk)	0	Check again

Base Conditions →

dp[i][j] = -1 (Not yet known).
0 (false).
1 (true)

Bottom Up Approach — H.W.

Doubly

x b b z z
x * z * * *

(x b b z z)
(x * z * * *)

x b b z
x * z * * *

x b b
x * z * *

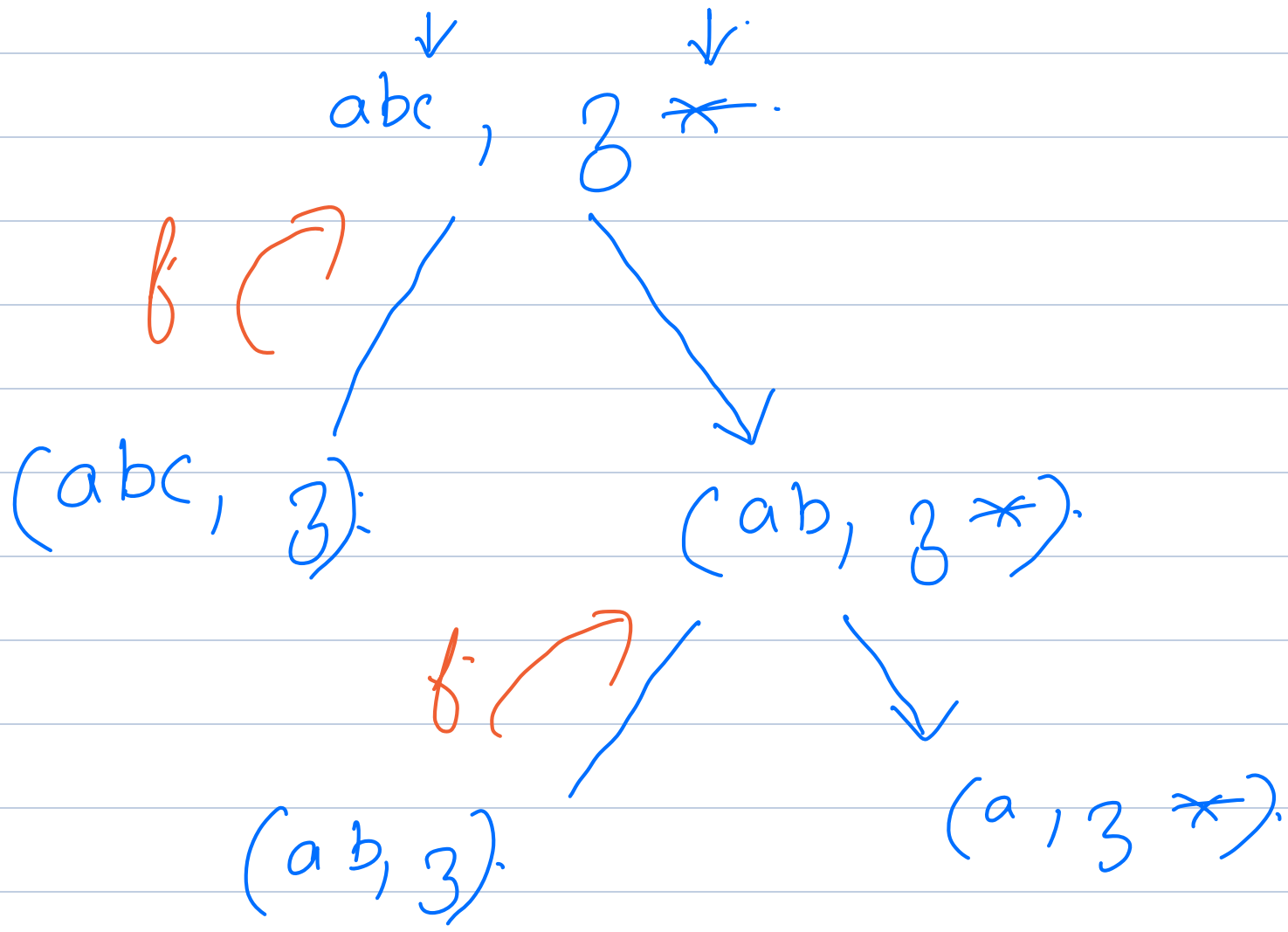
x b b z
x * z * * *

x b b
x * z * * *

x b
x * z * * *

$S_1 \rightarrow =$

$S_2 \rightarrow * * * *$


abc , 3*.
f
(abc, 3).
(ab, 3*).
f
(ab, 3).
(a, 3*).