Longest Common Subsequence

S1 = "adebc" S2 = "dcadb"

Brute force approch :- Generate all subsequence for S1 & S2.

S1 → $2^5 = 32$ → subsequence
S2 → $2^5 = 32$

And compare all the subsequence from S1 & S2 and find the LCS.

T.C. = exponential

Approch - 2:

> Generate all the subsequence and Compare on way

↳ standard way that we generaly do in DP on String

Generate all subsequence :- for this you use recursion where you use parameters that generate all subsequences.

Rules for recursion

(1) Express every thing in term of Index + find base case

how we have two strings so we use two diff. indexs so we can travers both the strings.

$f(idx1, idx2)$

(2) Explore all possibilities on that indexs.

(3) Take the Best among from all possibilities.

e.g. acd / ced

f(2,2) :- LCS of str1 [0 --- 2] & str2 [0 --- 2]

f(2,5) :- Give me the longest common subsequence b/w s1[0 --- 2] and s2[0 --- 5]

(2) Explore all possibilities

acd / ced
↑ idx1 ↑ idx2

Do comparision character-wise

(i) if matching → 1 + (ac / ce) → it means shrink the string and go for another matching in shrinked string

this mean we got 1 matched character

```
if( s1[idx1] == s2[idx2])
    1 + f(idx1-1, idx2-1)
```

(ii) NOT matched

~~ac / ce~~
~~idx1 idx2~~

ec / ce
idx1 idx2

idx1-1 → e / ce
idx1 idx2

idx2-1 → ec / c
idx1 idx2

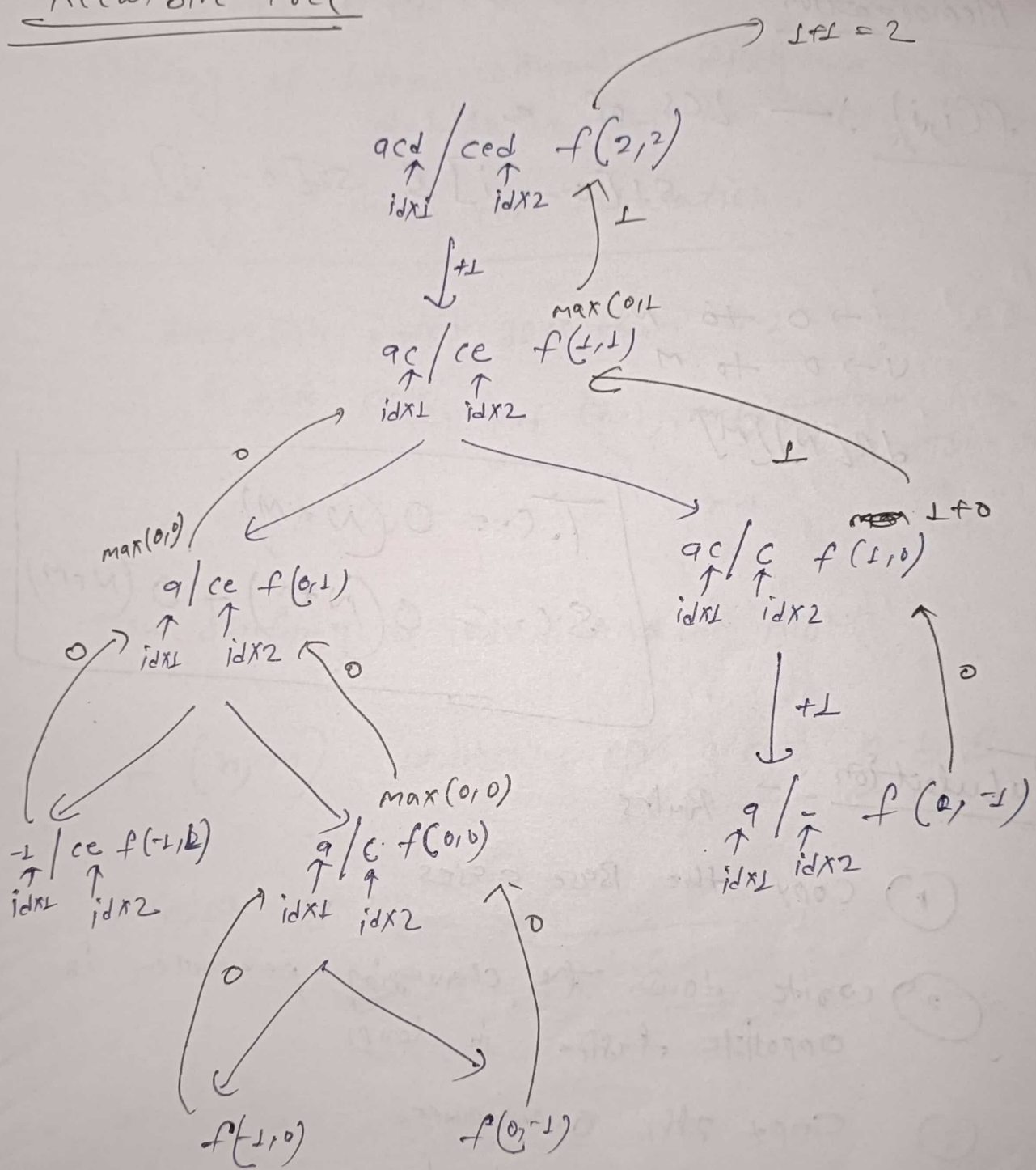
NotMatched = 0 + max(f(idx1-1, idx2), f(idx1, idx2-1));

```
f(idx1, idx2)
{
    if(idx1 < 0 || idx2 < 0)
        return 0;

    // If matched
    if(s1[idx1] == s2[idx2])
        return 1 + f(idx1-1, idx2-1);

    // if not matches
    return max(f(idx1-1, idx2), f(idx1, idx2-1));
}
```

~~Recursive Tree~~

→ It means the f^2 call return from here not go down further.

Recursive Trace



T.C. = $O(2^n * 2^m)$ → exponential

Memoization

f(i,j) :— LCS of a
s1[0---i] & s2[0--j]

i → 0 to N
j → 0 to M
dp[N][M]

T.C. = O(N*M)
S.C. = O(N*M) + O(N+M)

Tabulation :→ Rules

(1) Copy the Base cases

(2) write down the changing parameter in opposite fashion in loop

(3) Copy the recurrence.

⇒ Copy the Base case is bit tricy over here bcz

f(i<0 || j<0) return 0;

in above base case index can be -ve so we

can't write dp[-1][0] or so on. So we do shifting of index. without shifting we also do tabulation but that method is tricky.

Shifting of index for Tabulation :-

in Recursion index goes from n-1 to -1 & at -1 is base case f(n-1, m-1) ← from here recursion starts.

-1, 0, 1, 2, 3 ---- n-1

Now shifting of index to one Right

f(n,m) it means n means n-1 & m ——— m-1.

f(0,0) it mean 0 ——— -1 & 0 ——— -1

New base case

if(i==0 || j==0) return 0;

In tabulation

i=0 means dp[0][ ] = 0 (this can be anything)

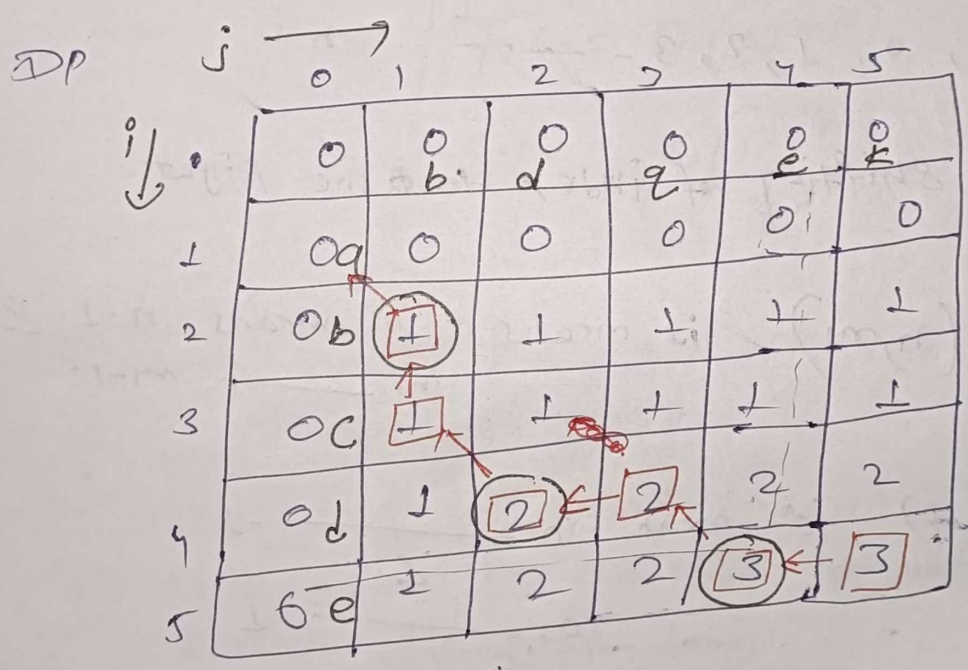
that means for(j=0 to n2) dp[0][j] = 0

j=0 means i can be anything

for(i=0 to n1) dp[i][0] = 0

Print LCS

text1 = "abcde" text2 = "bdqek"



LCS: b d e

dp[5][5] = 3

"abcde" & "bdqek" → LCS = 5 (b d e)

dp[4][2] = 2

"abcd" & "bd" → LCS = 2 (b d

//matched

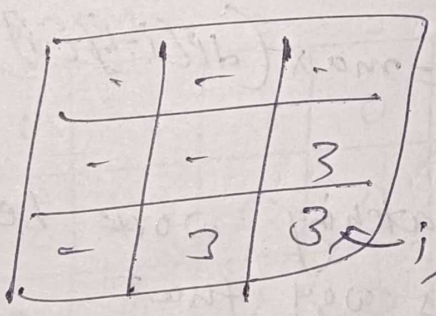
$text1[i-1] == text2[j-1]$

$dp[i][j] = 1 + dp[i-1][j-1]$

//not matched

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

~~Sup~~ Suppose



if($dp[i-1][j] == dp[i][j-1]$

that means both UP & Right have same value so we can move any where bcz same means there exist multiple answers