Lecture 13

$\boxed{\text{Longest Common Subsequence}}$

Given two sequences $A = (a_1\ a_2 \ldots a_n)$ and $B = (b_1\ b_2 \ldots b_m)$, find a Longest - Common - Subsequence (LCS) of A and B.

Eg. $A = (a\ c\ d\ b\ b\ c)$
$B = (c\ b\ d\ a\ c)$ $\}$ $LCS(A,B) = (c, d, c)$

Recursive algorithm

LCS(A, B, n, m):
    If (A[n] = B[m]) : Return LCS(A, B, n-1, m-1) • A[n]
    Else:
        ans 1 = LCS(A, B, n, m-1)
        ans 2 = LCS(A, B, n-1, m)
        If LENGTH(ans1) > LENGTH(ans2) : Return ans1
        Else : Return ans2

WILL INVOKE $\boxed{LCS(A, B, n-1, m-1)}$

Large time complexity

$$\text{Time} = O(2^{m+n})$$

Claim If $A[n] = B[m]$ then each $LCS(A,B)$ should end with

(Prove by contradiction)

$\underbrace{A\ P\ P\ L}\ \boxed{E}$

$\underbrace{C\ A\ P\ E}\ \boxed{E}$

New Solution

- Create 2-D array SIZE of dim $(n+1) * (m+1)$
- $\forall\, i \in [0,n],\ j \in [0,n]$ if $i=0$ or $j=0$ $SIZE[i,j] = 0$

  else $SIZE[i,j] = -1$

Goal: Store in SIZE[i,j] length of LCS(A[1,i]

LCS(A, B, n, m):

If (A[n] = B[m]):

If SIZE[n-1, m-1] = -1 then invoke LCS(A,B,n-1,m-1)

SIZE[n,m] = SIZE[n-1, m-1] + 1

Else:

If SIZE[n-1, m] = -1 then invoke LCS(A,B,n-1,m)

If SIZE[n, m-1] = -1 then invoke LCS(A,B,n,m-1)

SIZE[n,m] = max (SIZE[n-1, m], SIZE[n, m-1])

Print_LCS(i,j)

If (A[i] = B[j]) { abbended A[i] to LCS(i-1, j-1)

Print_LCS(i-1, j-1) • A[i]

Back track

Else If SIZE[i,j] = SIZE[i-1,j]

Print_LCS(i-1, j)

Else

Print_LCS(i, j-1)

(Back-tracking to find the LCS)

Time = O(m + n)

Ques. If SIZE[i,j] = 1 + SIZE[i-1, j-1] then A[i] = B[j]

No

a b c

a c b

SIZE(3,3) = 2

$\boxed{\text{EDIT DISTANCE PROBLEM}}$

Given two strings $A = (a_1 .. a_n)$ and $B = (b_1 .. b_m)$, convert A to B by following o/ps:

* Remove(i)
* Insert(x, i)
* Replace(x, i)

Edit Distance (A, B) = minimum number of operations needed to go from A to B.

dist (BAT HAT) = 1   Last is same

dist (BAT HATS) = 2   Append S   1 + dist (BAT HAT)

dist (BAN HAT) = 2   Replace N→T   1 + dist (BA HA)

dist (BANK HAT) = 3   Delete last $1 + \underbrace{\text{dist (BAN HAT)}}_{\text{subproblems}}$

- $A[n] \neq B[m]$

$$
\text{MIN}
\begin{cases}
\begin{cases}
A[1] \ldots A[n-1] \quad A[n] \quad B[m] \\
B[1] \ldots B[m-1] \quad B[m] \\
1 + \text{dist}(A[1,n],\ B[1,m-1])
\end{cases}
\\[2em]
\begin{cases}
A[1] \ldots A[n-1] \quad \cancel{A[n]}\ B[m] \\
B[1] \ldots B[m-1] \quad B[m] \\
1 + \text{dist}(A[1,n-1],\ B[1,m-1])
\end{cases}
\\[2em]
\begin{cases}
A[1] \ldots A[n-1] \quad \cancel{A[n]} \\
B[1] \ldots B[m-1] \quad B[m] \\
1 + \text{dist}(A[1,n-1],\ B[1,m])
\end{cases}
\end{cases}
$$

Algorithm

1. Initialize 2-D array "dist" of size $(n+1) \times (m+1)$.
2. For $i = 0$ to n $\quad dist[i, 0] = i$

   For $j = 0$ to m $\quad dist[0, j] = j$
3. For $i = 1$ to n:

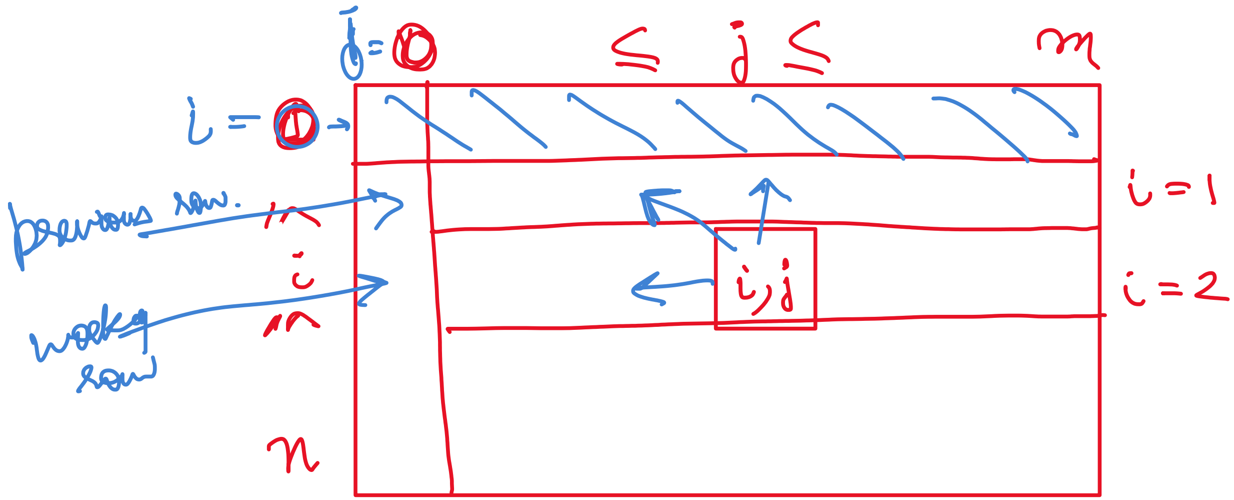
   For $j = 1$ to m:

   If $(A[i] = B[j])$: $dist[i, j] = dist[i-1, j-1]$

   Else $dist[i,j] = 1 + \min \begin{cases} dist[i][j-1] & \text{\# Insert } B[j\ldots \\ dist[i-1][j-1] & \text{\# Replace to } B\ldots \\ dist[i-1][j] & \text{\# Remove } A[\ldots \end{cases}$
4. Return $dist[n][m]$.

current sol^n: space, time $= O(mn)$

CLAIM Space can be reduced to $O(n)$ if we are interested in



H.W. If you require to know the complete sequence of operations then can you have an algo with O(n + m) space

In O(m * n) space it is easy

A[1]
GOLDEN
MOLDEN ← 1

Replace A[1] with B[1]
G → M

MODEN ← (2) Remove A[3]

← (3) Insert R at location 5

MODERN

B[1]