CS & IT ENGINEERING

Algorithm

Dynamic Programming

Lecture No.- 04

By- Dr. Khaleel Khan sir

Recap of Previous Lecture

Topic All Pairs Shortest Paths

Topic 0/1 Knapsack

LCS

Topics to be Covered

Topic L.C.S ✓

Topic Matrix Chain Product (MCP) ✓

S.o.S ✓

0/1 KNAP ✓

Flyod-Warshall's

Topic : Dynamic Programming: (DP)

Longest Common Subsequence (LCS)

Given Two Strings X & Y of length 'n' & 'm' charac's, a Subsequence that is common to both 'X' & 'Y' is known as Common Subsequence;

$X = \langle A\,B\,C\,D \rangle \quad Y = \langle B\,D\,C \rangle$

$\langle A \rangle$ ✗ $\quad \langle B \rangle$ ✓ $= 1$

✓ $\langle B C \rangle$ ✓ $= 2$

$\langle B C D \rangle$ ✗

Given Strings X & Y of length 'n' & 'm' char's, the Problem of LCS is to determine a Subsequence of <u>longest length</u> that is Common to both 'X' & Y;

⟹ (Realistic Algo)

Appl's:

1) Genomics: Genetic Engg: for DNA Strands Sampling;

$X = \langle \underline{A\ B\ C\ B\ D\ A} \rangle$; $Y = \langle Y\ X\ D\ B\ C\ A\ D\ K \rangle$

$'X' = \langle value \rangle$

2) Version change: in S/W Engg.

3) Data gathering Systems: (Search Engines)

4) Plagiarism: (Research) ⟨Ravi ate an apple⟩

(Apple was eaten by Ravi)*

1) $X = \langle CGATAATTGAGA \rangle$

$\langle CTAAT \rangle$ ✓

$\langle TTGAC \rangle$ ✗

2) $X = \langle ABCBDAB \rangle$, $Y = \langle BDCABA \rangle$

$\underline{\underline{\lambda}} = 4$

$\langle BCB \rangle$ ✓ $= 3$

$\langle AB \rangle$ ✓ $= 2$

$\langle BCBA \rangle$ ✓ $= 4$

$\langle BCAB \rangle$ ✓ $= 4$

D.P based Soln for L.C.S:

Let 'i' & 'j' be indices into the Strings 'X' & 'Y' as Shown

$$X = \langle \overset{0}{x_1}, \overset{1}{x_2}\, \overset{2}{x_3} \cdots \overset{i}{x_n} \rangle \;;\; Y = \langle \overset{0}{y_1}\, \overset{1}{y_2}\, \overset{2}{y_3} \cdots \overset{j}{y_m} \rangle$$

→ Let $L(i, j)$ denote the length of Common Subsequence of the strings X & Y as defined above;

$$L(i,j) = 1 + L(i-1, j-1) \;,\; \text{if } X[i] = Y[j]$$

$$= \max\{L(i-1, j),\; L(i, j-1)\} \;,\; \text{if } X[i] \neq Y[j]$$

$L[-1, j] = 0$, for $i = -1$, $j = -1, 0, 1, 2, \ldots$

$L[i, -1] = 0$, for $j = -1$, $i = -1, 0, 1, 2, \ldots$

Case I: $X = \langle \overset{0\,1\,2\,3\,4\,5\,6\,7\,8\,9}{GTTCCTAATA} \rangle$

$Y = \langle \underset{0\,1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11}{CGATAATTGAGA} \rangle$

$L(9,11)$

$X[i] = Y[j]$

$$L(i,j) = 1 + L(i-1, j-1)$$

Case II:

$X = \langle \overset{0\,1\,2\,3\,4\,5\,6\,7\,8\,9}{GTTCCTAATA} \rangle$ (i = 9)

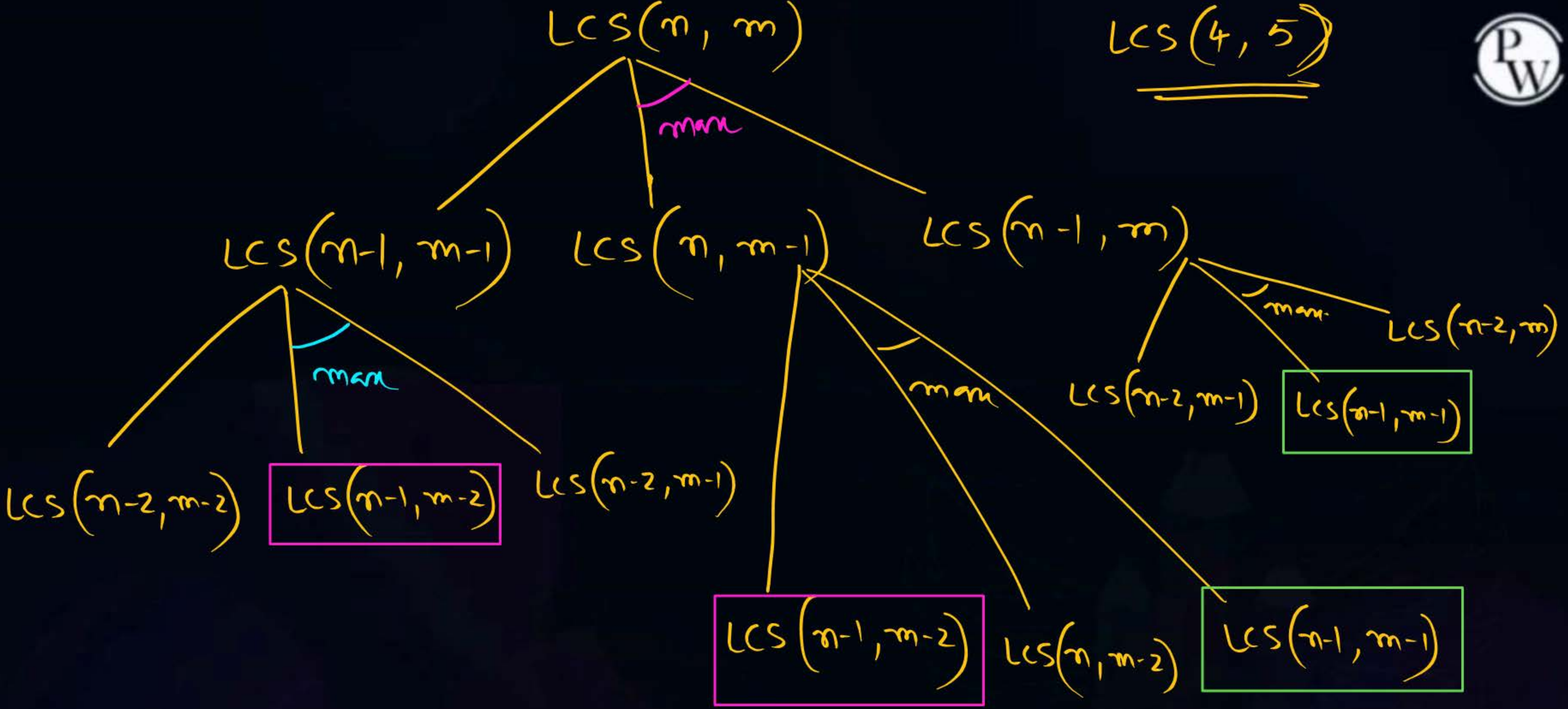
$Y = \langle \underset{0\,1\,2\,3\,4\,5\,6\,7\,8\,9\,10}{CGATAATTGAG} \rangle$ (j = 10)

$L(9,10) =$

$X[i] \neq Y[j]$

$$L[i,j] = \max\{L(i-1, j),\ L(i, j-1)\}$$

LCS(4, 5)



LCS(n, m)
max
LCS(n-1, m-1)
LCS(n, m-1)
LCS(n-1, m)
max
LCS(n-2, m-2)
LCS(n-1, m-2)
LCS(n-2, m-1)
max
LCS(n-1, m-2)
LCS(n, m-2)
LCS(n-1, m-1)
max
LCS(n-2, m-1)
LCS(n-1, m-1)
LCS(n-2, m)

1. Recursion Tree :

LCS("ABBAB", "ACBAB") = 4

1 + B

LCS("ABBA", "ACBA") = 3

1 + A

LCS("ABB", "ACB") = 2

1 + B

LCS("AB", "AC") = 1

max

A　　　　A

1 = LCS("AB", "A")　　　LCS("A", "AC") = 1

max

0 = LCS("AB", "")　　LCS("A", "A") = 1　　1 = LCS("A", "A")　　0 LCS("", "AC") = 0

1 +　　　　1 +

LCS("", "") = 0　　LCS("", "") = 0

"ABAB"

AB

$X = \langle A\ B\ C\ B\ D\ A\ B \rangle$; $Y = \langle B\ D\ C\ A\ B\ A \rangle$

$\langle AB \rangle$ $\langle BDCAB \rangle$

$AB = 2$

L

	y_i	-1	0: B	1: D	2: C	3: A	4: B	5: A
-1		0	0	0	0	0	0	0
0	A	0	0 $\uparrow$	0 $\uparrow$	0 $\uparrow$	$\nwarrow 1_A$	$\leftarrow 1$	$\nwarrow 1$
1	B	0	$\nwarrow 1_B$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 2_B$	$\leftarrow 2$
2	C	0	1 $\uparrow$	1 $\uparrow$	$\nwarrow 2$	$\leftarrow 2$	2 $\uparrow$	2 $\uparrow$
3	B	0						
4	D	0						
5	A	0						
6	B	0						$\leftarrow 4$

x_i

$X = \langle ABC \rangle$ $Y = \langle B \rangle$

$\langle B \rangle = 1$

$1 + L[i-1, j-1]$

$L[1, 4)$

$L[0, 3] = 1$

(AB)

Topic : Algorithms

Algorithm LCS based on Bottom-up Tabulation method:

```
Algorithm LCS (x, y)
integer x [ 0..n] , y[0..m];
{
        integer L [ 0..n , 0..m];
1.      for i ← 0 to n –1
            L [i– 1] = 0;
2.      for j ← 0 to m –1
            L [–1, j]  = 0;
3.      for i ← 0 to n –1
            for j ←0 to m –1
                if (x [i] = y [j]) then
                    L [I , j] = 1 +  L [i - 1, j –1];
                else
                    L [i, j] = max {L[i , j –1], L [i –1, j]}
}
```

Boundary Condition

Time : O(n * m)

Space: O(n * m)

Slide 5

6) Matrix-chain Product (MCP)

1) Given Two Square Matrices A & B of order $n \times n$;

The no. of Scalar Multiplications needed to multiply them $(A \times B) = \underline{\quad n^3 \quad}$,

$$A \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}_{\substack{2\times 2 \\ n\times n}} \quad B \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}_{\substack{2\times 2 \\ n\times n}} = C \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$\left.\begin{aligned} c_{11} &= a_{11}\cdot b_{11} + a_{12}\cdot b_{21} \\ c_{12} &= a_{11}\cdot b_{12} + a_{12}\cdot b_{22} \\ c_{21} &= a_{21}\cdot b_{11} + a_{22}\cdot b_{21} \\ c_{22} &= a_{21}\cdot b_{12} + a_{22}\cdot b_{22} \end{aligned}\right\} 2\times 2\times 2 = 8$$

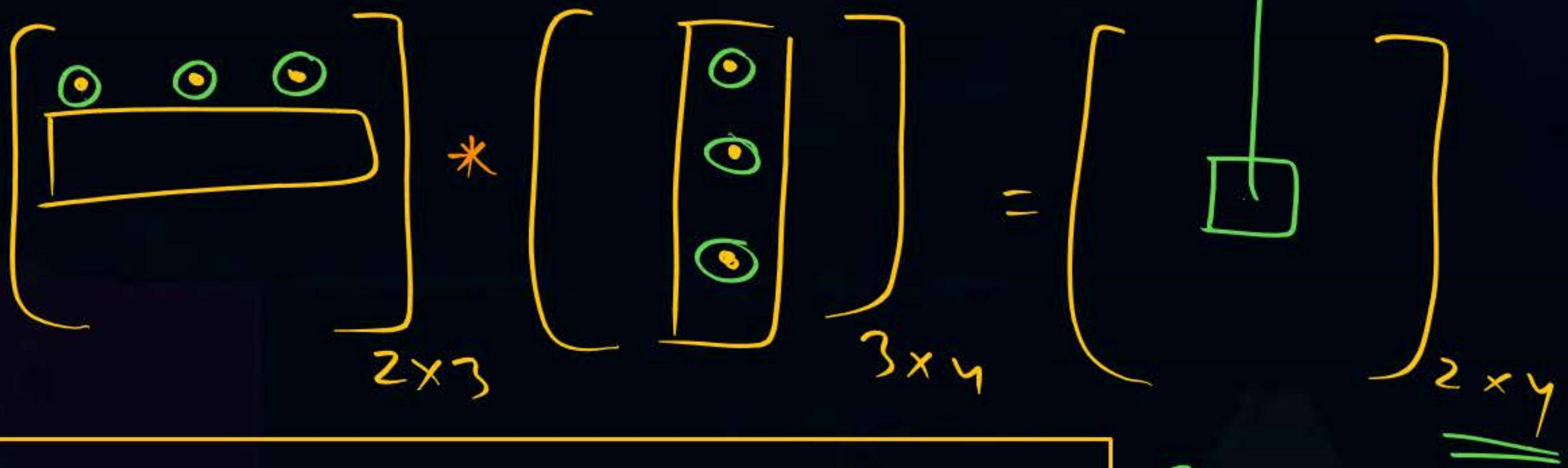
for $i \leftarrow 1$ to n

for $j \leftarrow 1$ to n

for $k \leftarrow 1$ to n

2) Non-Square Matrix Multiplication:

$A_{2\times \underline{3}} * B_{\underline{3}\times 4} = C_{2\times 4} \Rightarrow$ **Compatible** 3



$(2*4)*3 = (2\times 4\times 3)$

$$A_{n\times m} * B_{m\times k} = C_{n\times k}$$

Total Scalar Multiplication $= n \times m \times k$

$\Longrightarrow$ Given a chain of Compatible Non-Square Matrices, it is reqd. to Multiply them together to get one Final Matrix;

Ex: $\langle B_{2\times10} ; C_{10\times50} ; D_{50\times20} \rangle = \langle BCD \rangle = A_{2\times20}$

(No. of Scalar Multiplications)

$\langle BCD \rangle$

I: $\left[\underbrace{(B\cdot C)}_{2\times50} \cdot D \right]$

$(2\cdot10\cdot50) + (2\cdot50\cdot20)$

$1000 + 2000 = 3000$

II: $\left[\left(\underset{2\times10}{B} \cdot \underset{10\times20}{(C\cdot D)} \right) \right]$

$2\cdot10\cdot20 + (10\cdot50\cdot20)$

$400 + 10{,}000 = 10{,}400$

Topic : Algorithms

Algorithm Matrix-Chain-Product (p)

```
n ←length[p] — 1
for i ← 1 to n
        do m[i, i] ← 0
for l ← 2 to n      l is the chain length.
  for i ← 1 to n - l + 1
        j ← i + l - 1
             m[i, j]← ∞
             for k ← i to j - 1
                  q ← m[i, k] + m[k + 1, j] + P_{i-1}P_kP_j
                       if (q < m[i, j] )
                            then m[i, j] ← q
                                 s[i, j] ← k
return m and s
```

SOS can be implemented using bottom-up DP with Tabulation:

Algorithm SOS (n , M, A)

```
            A [1.....n]
{
  integer X[0..n , 0..M];

1.    for i ← 0 to n
        for j ← 0 to M
          if (i ≥ 0 and j = 0)
             X [i , j]= T
          else
             if (i = 0 and j > 0)
              X [i, j] = F;
             else
                if (A [i] > j )
                   X [i,j] = X [i –1, j]
                else
                   X [i, j] = X [i–1, j] V X [i–1, j –A [i] ]

}
```

Topic : Algorithms

Algorithm Bellman-Ford (G, w, s)

1. Initialize-Single-Source(G,s)
2. for i ← 1 to |V[G]| − 1
3.   do for each edge (u, v) ∈ E[G]
4.   do Relax(u, v, w)
5. for each edge (u, v) ∈ E[G]
6.   do if d[u] > d[w] + w(u, v)
7.   then return FALSE
8. return TRUE

Topic : Algorithms

Initialize-Single-Source (G, s)

```
for each vertex v ∈ V[G]
        do d[u] ← ∞
              π[v] ← NIL
d[s] ← 0
```

Relax (u, , w)

```
if d[v] > d[u] + w(u, v)
        then d[v] ← d[m] + w(w, u)
              π[u] ← u
```

THANK - YOU