CS & IT ENGINEERING

Algorithms

Dynamic Programming (DP)

Lecture No.- 03

By- Aditya Jain sir

Recap of Previous Lecture

Topic

Topic

DP Strategies

SSSP

APSP

Topics to be Covered

Topic

Topic 0/1 Knapsack

Topic LCS

About Aditya Jain sir

1. Appeared for GATE during BTech and secured AIR 60 in GATE in very first attempt - City topper
2. Represented college as the first Google DSC Ambassador.
3. The only student from the batch to secure an internship at Amazon. (9+ CGPA)
4. Had offer from IIT Bombay and IISc Bangalore to join the Masters program
5. Joined IIT Bombay for my 2 year Masters program, specialization in Data Science
6. Published multiple research papers in well known conferences along with the team
7. Received the prestigious excellence in Research award from IIT Bombay for my Masters thesis
8. Completed my Masters with an overall GPA of 9.36/10
9. Joined Dream11 as a Data Scientist
10. Have mentored 12,000+ students & working professions in field of Data Science and Analytics
11. Have been mentoring & teaching GATE aspirants to secure a great rank in limited time
12. Have got around 27.5K followers on Linkedin where I share my insights and guide students and professionals.

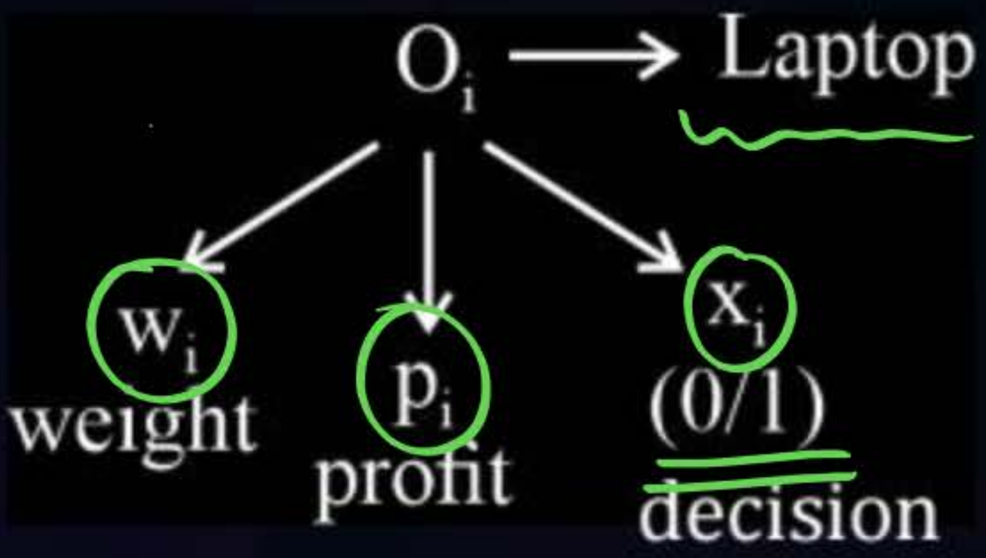
@ADITYASIR_PW

Telegram Link for Aditya Jain sir: https://t.me/AdityaSir_PW

Topic : Dynamic Programming

0/1 Knapsack (Binary Knapsack):

- Knapsack Capacity : M
- no. of objects → n



Objective Function:

$$\max \sum_{i=1}^{n} p_i * x_i$$

Such that:

$$\Sigma w_i * x_i \leq M$$

$$x_i = \{0, 1\}$$

Dynamic Programming Approach:

- Let 01 Knap(n, M) represent profit with n-objects and Knapsack of Capacity M.

Topic : Dynamic Programming

V.imp

Recurrence:

$$01\ \text{Knap}(n, M) = 01\ \text{Knap}(n-1, M),\ w_n > M \quad \rightarrow \text{Exclude } O_n \text{ (no option)}$$

$$01\ \text{Knap}(n, M) = \max\begin{cases} 01\ \text{Knap}(n-1, M) \\ 01\ \text{Knap}(n-1, M-w_n) + p_n \end{cases},\ w_n \leq M$$

Initialization,

$$01\ \text{Knap}(n, M) = 0, \text{ if } n = 0 \text{ or } M = 0$$

TD

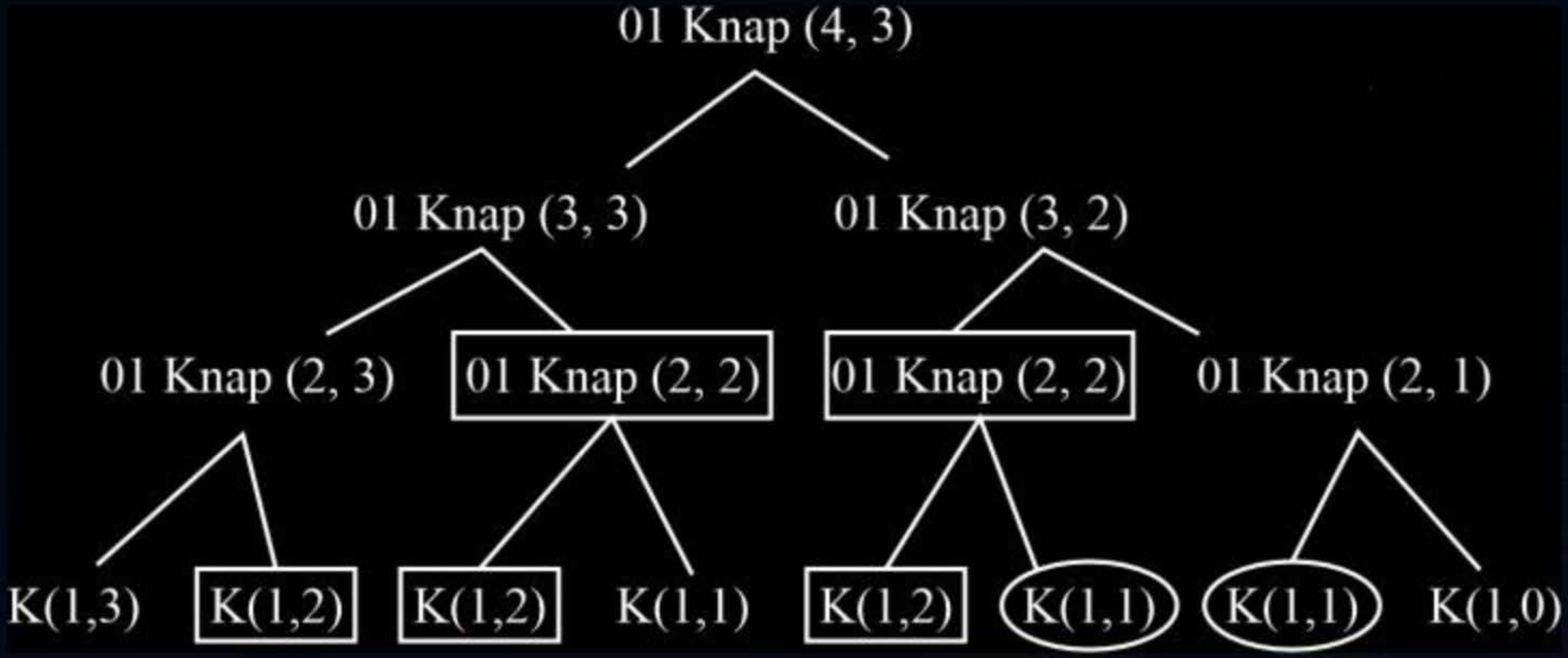
Example:

To understand if there are overlapping Sub-problem or not?

$n = 4, M = 3$

	01	02	03	04
P_i profits	10	20	30	40
W_i weights	1	1	1	1

Top-Down (Recursive):



Topic : Dynamic Programming

Bottom-up Approach (Tabulation):

Example:

$n = 4, M = 8$

	01	02	03	04
P_i	1	2	5	6
W_i	2	3	4	5

Tabulation:

Let $X[0 \ldots n, 0 \ldots M]$ be an array of which $X[n, M] \Rightarrow$ final answer (max profit)

Topic : Dynamic Programming

P_i	W_i
1	2
2	3
5	4
6	5

m = 8

HW

m

n

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

KS(2, 3)

Maximum Profit

Topic : Dynamic Programming

$X[i, j] = \max\{X[i-1, j], X[i-1, j-W_i] + P_i\}$

$X[2, 3] = \max\{X[1, 3], X[1, 3-3] + 2\}$

$X[2, 3] = \max\{1, 0+2\} = \max[1, 2] = 2$

$X[2, 5]$ = max profit we can have with considering only 1^{st} two objects and knapsack of capacity 5.

Topic : 0/1 Knapsack Code

```
Algo 01Knap(M, n, W, P) {
    X[0...n, 0...M]
    for (i = 0; i <=n; i++) {
      for (j = 0; j <= M; j++) {
            if (i == 0 or j == 0) {
                X[i, j] = 0
            } else if (W[i] ≤ j) {
                X[i, j] = max(X[i−1, j], X[i−1, j−W[i]] + P[i])
            } else {
                X[i, j] = X[i−1, j]
            }
        }
    }
    return X[n, M]
}
```

Topic : 0/1 Knapsack Code

X	0	1 2 . . .	M	
0				
. . . .				
n				

X[n, M]

Topic : 0/1 Knapsack Code

#Q. $n = 3, M = 6$

	01	02	03
P_i	1	2	5
W_i	2	3	4

Topic : Dynamic Programming

Complexity Analysis of 0/1 Knapsack:

1. Time Complexity:

   - $O(n * M)$

2. Space Complexity:

   - $O((n+1) * (M+1)) = O(n * M)$

Topic : Dynamic Programming :(DP)

6. Sum of subset: (SOS)

Problem Statement:- Given a set of n-elements (integers) and also another elements (integer) 'M'.

The sum of subsets (SOS) problem is to determine If there exits a subset of the given elements whose sum equals to 'M'.

SOS→ Decision Problem → o/p: True / False

Topic : Dynamic Programming :(DP)

DP Problems So far:

1. Bellman – Ford→ Minimize path cost (SSSP)
2. Floyd Warshall → Minimize path cost (APSP)
3. 0/1 knapsack → Maximize Profit
4. Longest common Subsequence (LCS) → Maximize length of common subsequence.
5. Matrix chain multiplication (MCM) → Minimize scalar multiplication

Topic : Dynamic Programming :(DP)

SOS→ Very similar to 0/1 knapsack problem:

Eg.1. N = 5

A =[50, 30, 10, 40, 20]

M = 50

Subsets	Index
{50}	{1}
{30, 20}	{2, 5}
{10, 40}	{3, 4}

Topic : Dynamic Programming :(DP)

Enumeration

Brute Force:-

Check every possible subsets

n elements $\longrightarrow$ $2^n \rightarrow$ subset

TC: $O(2^n)$

Topic : Dynamic Programming :(DP)

DP: SOS→ Top-Down (Recurrence of SOS DP)

Target Sum

↑

n, A[Ai…… An], M,

Let SOS (n, m) represent the solution to the SOS, with n elements and target sum (M).

SOS (n, m) = True/ False = 0/1

Topic : Dynamic Programming :(DP)

SOS (n, m) = T {There exits a subset from n elements that sum to M}

= F { There does not exits}

OR

F or F → F ✓

F or T → T

T or F → T

T or T → T

Topic : Dynamic Programming :(DP)

V.imp

$$SOS(n, M) = SOS(n-1, M)\ ,\ \underline{An > M}\ \ \{\text{skip/ reject An}\}$$

$$SOS(n, M) = \left\{ \begin{array}{c} SOS(n-1, M) \\ \text{or} \\ SOS(n-1, \underline{M - An}) \end{array} \right\}, \ \underline{An \le M}$$

It we can get to sum M from either taking An or Rejecting An

Choice

taking reject

Topic : Dynamic Programming :(DP)

Boundary case

or

Base condition

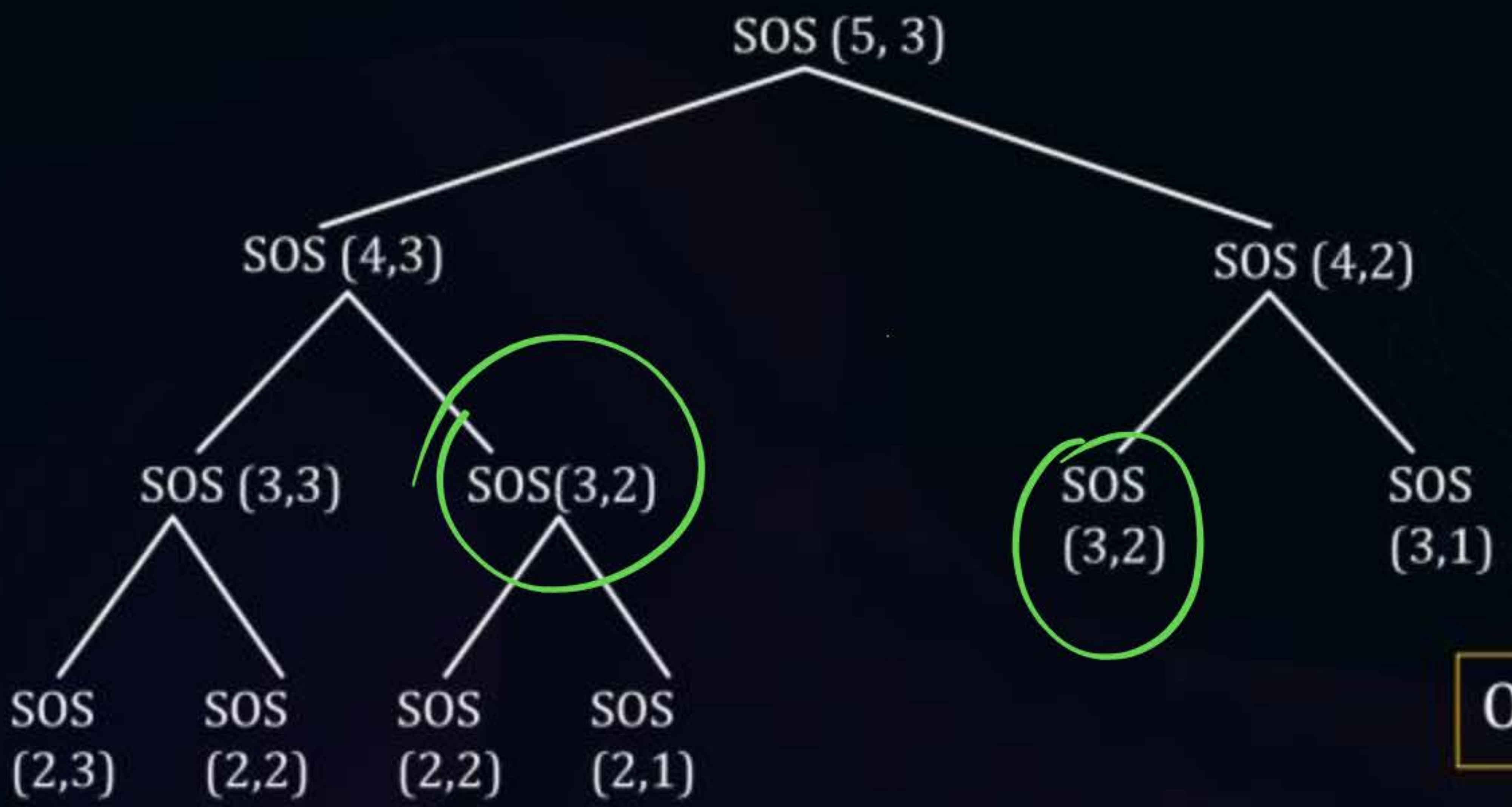
SOS (n, m) = False , n = 0 & M > 0

SOS (n, m) = True , M = 0 & n ≥ 0

Topic : Dynamic Programming :(DP)

TOP – Down

E.g.:- $n = 5$, $A = [1, 1, 1, 1, 1]$ $M = 3$



Overlapping subproblems

Topic : Dynamic Programming :(DP)

SOS → Bottom Up (Tabulation Method)

Given:- n, M   x [0, ...n , 0.....M]

X[i,j] = True /False

↓

A particular i,j call form x[,]

Whether there exits a subset from the 'i' elements that sum to 'j'.

E.g.2. A = [2, 8, 4, 11, 9]

n= 5,

m = 6

Topic : Dynamic Programming :(DP)

M →

Ai	X	0	1	2	3	4	5	6
	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
8	2	T	F	T	F	F	F	F
4	3	T	F	T	F	T	~~T~~ F	T
11	4	T	F	T	F	T	~~T~~ F	T
9	5	T	F	T	F	T	~~T~~ F	T

↓

Final Answer

Topic : Dynamic Programming :(DP)

Imp ~~Shortest~~ Shortcut

In A [i , j] = True then the entire column 'j' can be set to True.

Topic : Dynamic Programming :(DP)

DP: SOS: Tabulation (Bottom -up) Code

```
Algo SOS (n, M, A)
            A [1..........n]
{
  X [0....n,   0.......M]   → Auxiliary Space
            for i = (0 to n):
  {
            for j = (0 to M)
            {
                    if (i ≥ 0 and j =0)
                    {
                            X [i, j] = True;
                    }
```

Topic : Dynamic Programming :(DP)

```
else if [i = 0 and j > 0]
    x [i,j] = False;
}
else if (A [i] > j)
{
        x [i,j] = x [i-1 , j]
}
else
{
    x [i,j] = (x[i-1, j]) or (x[i-1, j -A [i]])
}
}
}
}
```

Topic : Dynamic Programming :(DP)

Complexity Analysis of Bottom- up (DP) Approach of SOS:

1. Time complexity : O(n*M)
2. Space complexity : O(n*M)

Note:-

Longest Common Sub-Sequence (LCS):

- Based on Strings
- A sequence/group/array of one or more characters.

String:

1) Substring:

- A contiguous sequence of 1 or more characters from the given string.

Topic : Dynamic Programming

String:

2) Subsequence:

- A sequence of 1 or more characters from the given string, that may or may not be contiguous, but their relative order is maintained.

Topic : Dynamic Programming

Example:

- String: "ABC" → length = 3
- Sub-strings: A, B, C → 3

  AB, BC → 2

  ABC → 1

No. of substrings = 3 + 2 +1 = 6

"ABC"

Important Result:

Given : S[1, ..., n]

#Q. What are the total number of sub-strings of at-least 1 length (non-empty sub-strings)?

$\underline{\underline{n=4}}$

$\frac{n(n+1)}{2}$

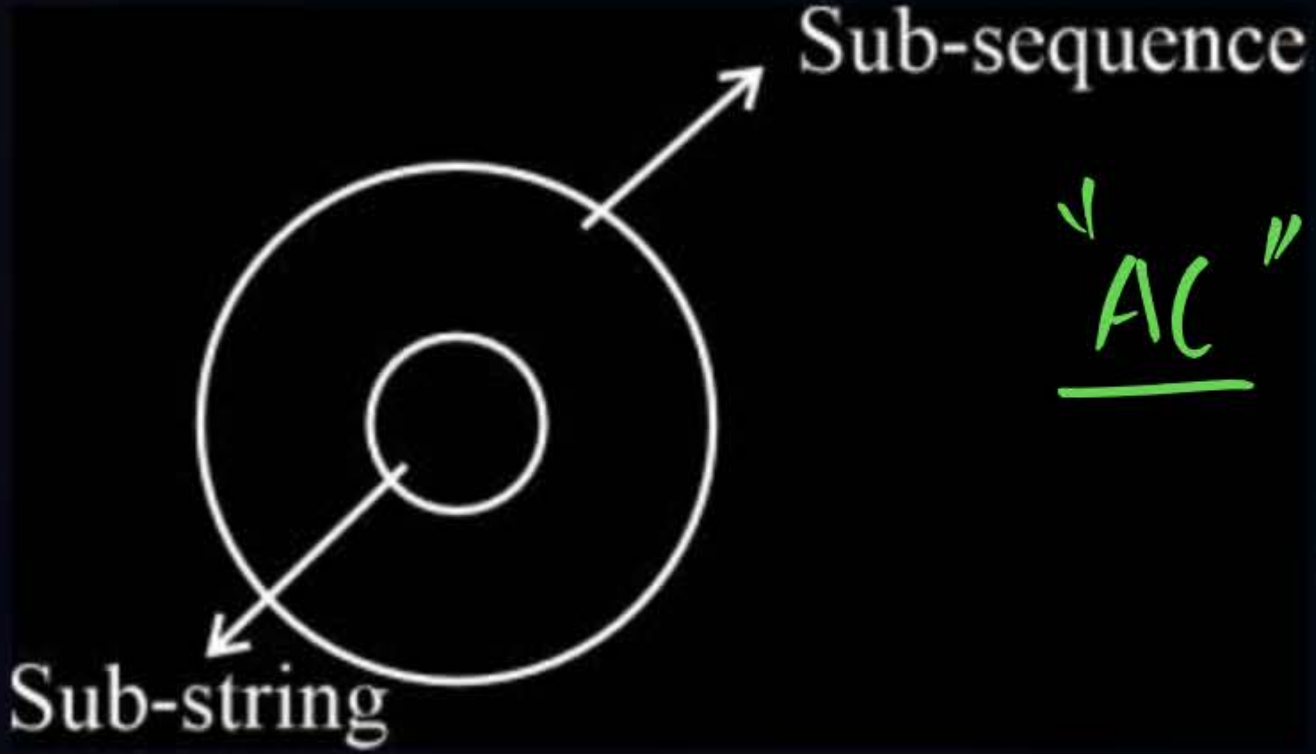
Topic : Dynamic Programming

Sub-Sequence:

- String = "ABC"
- Sub-Sequence = A, B, C, AB, BC, AC, ABC → Valid sub-sequence
- CA, CB → relative order not maintained, not a sub-sequence.

Important:

- Every substring is also a sub-sequence, but a Sub-sequence may or may not be a substring.



"AC"

[MSQ]

#Q. Given a string S = "CABDABBCD".

Which of the following are valid sub-sequences?

A BBD

B BAC

C CDCA

D CADC

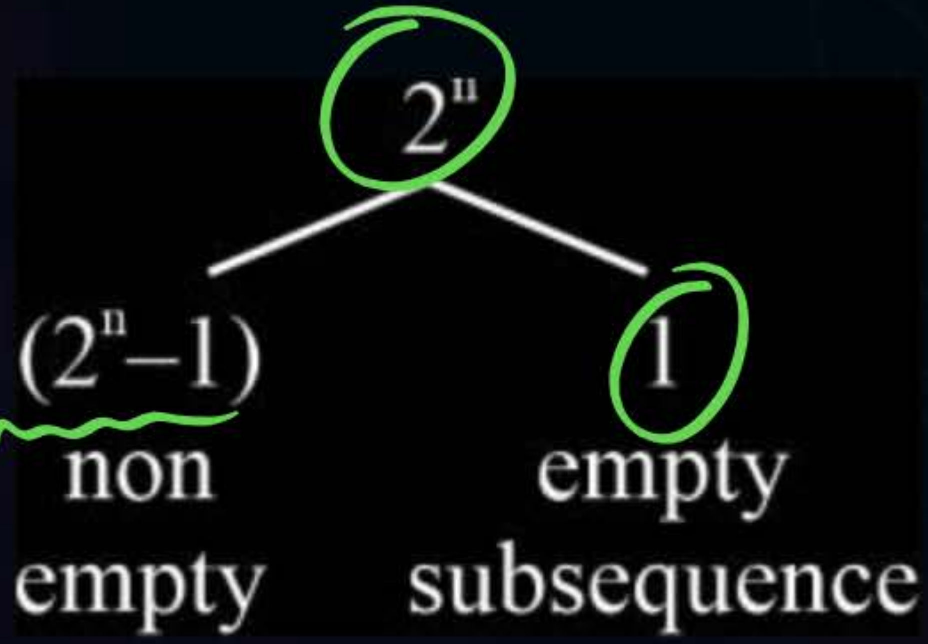
E ADABD

Longest Common Sub-Sequence:

#Q. Given a string of length 'n' characters, the number of sub-sequences possible are?

Longest Common Sub-Sequence:

$\varphi \rightarrow$ empty sub-sequence



Topic : Dynamic Programming

Given

$$\left.\begin{array}{l} \text{'n' All Subsequence} \rightarrow A \\ \text{'n' All Subsequence} \rightarrow B \end{array}\right\} \text{Common} \rightarrow \text{longest}$$

- Brute Force/Enumeration: $O(2^n)$
- Hence, we need DP based approach.

Longest Common Subsequence (DP based approach):

Common Subsequence:

- Given two strings X and Y of length n and m respectively, a subsequence that is common to both X and Y is known as a common sub-sequence.

Topic : Dynamic Programming

Example:

#Q. X = "ABCD", Y = "BDC"

Which sub-sequences are common to both?

1) A
2) B
3) BC
4) BD
5) BCD
6) DC
7) BDC

LCS Problem Statement:

#Q. Given strings 'A' and 'B' of length (Characters) 'n' & 'M' respectively. The LCS problem is to determine the subsequence that is of the longest length among all the subsequences that are common to both 'A' & 'B'?

Example:

S1 = "ABCBDAB"

S2 = "BDCABA"

Longest LCS =

Common Subsequences:

AB → 2

BCA → 3

BCB → 3

BCBA → 4

BCAB → 4

- Multiple longest common subsequences are possible.
- But LCS length is unique → 4

DP based solution to LCS:

- Let 'i' and 'j' be indices in the strings 'X' and 'Y' respectively, such that.

$X = [x_1, x_2, x_3, \ldots, x_n]$

$Y = [y_1, y_2, y_3, \ldots, y_m]$

Topic : Dynamic Programming

- Let L(i, j) represent the length of the common sub-sequence of the strings X and Y.

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

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

X

Y

Topic : Dynamic Programming

Initialization:

- $L[0, j] = 0,$

  1^{st} string is exhausted

  if $i = 0; j = 0, 1, 2, \ldots, m$

- $L[i, 0] = 0,$

  2^{nd} string is exhausted

  if $j = 0; i = 0, 1, 2, \ldots, n$

Topic : Dynamic Programming

Example:

Case1: X[i] = Y[j]

X = "ABBCDCCABA"

Y = "CBDADCBDAA"

Topic : Dynamic Programming

Example:

Case2: $X[i] \neq Y[j]$

X = "ABCDBBCCADB"

Y = "BDBCAAABDCD"

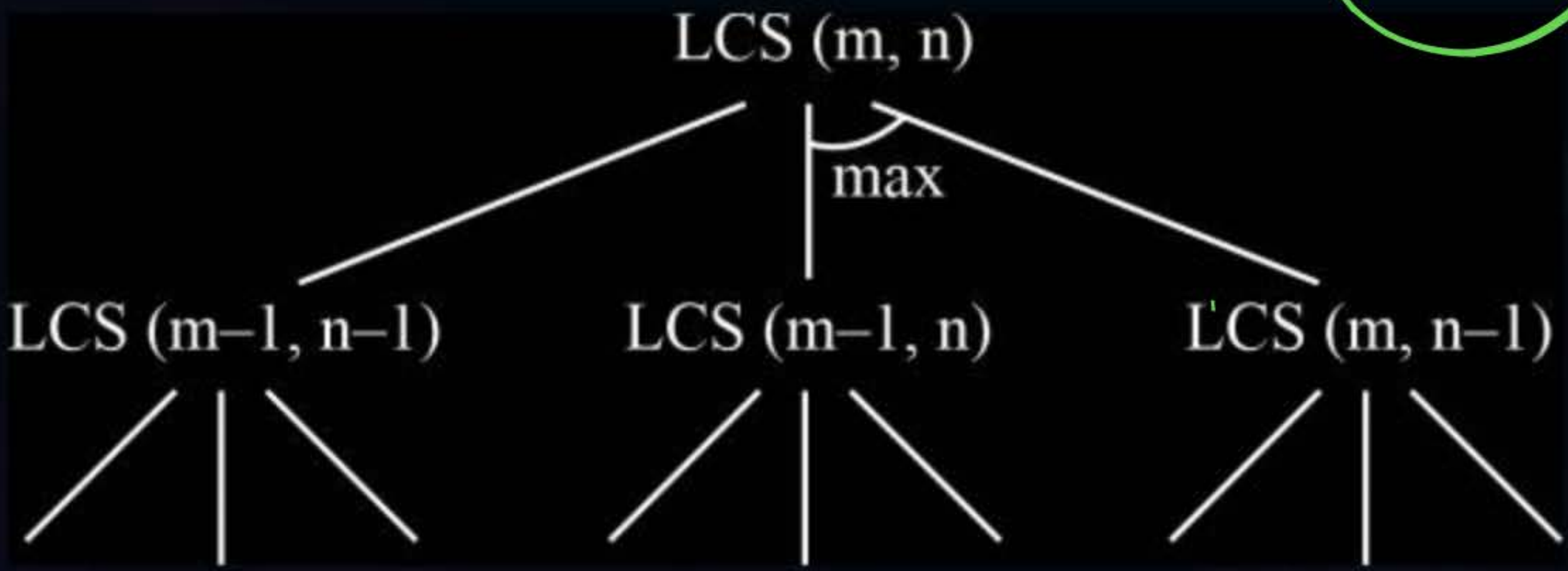
$\max\{L(i, j-1), L(i-1, j)\}$

Topic : Dynamic Programming

Overlapping Subproblems:

$|X| = n, |Y| = m$

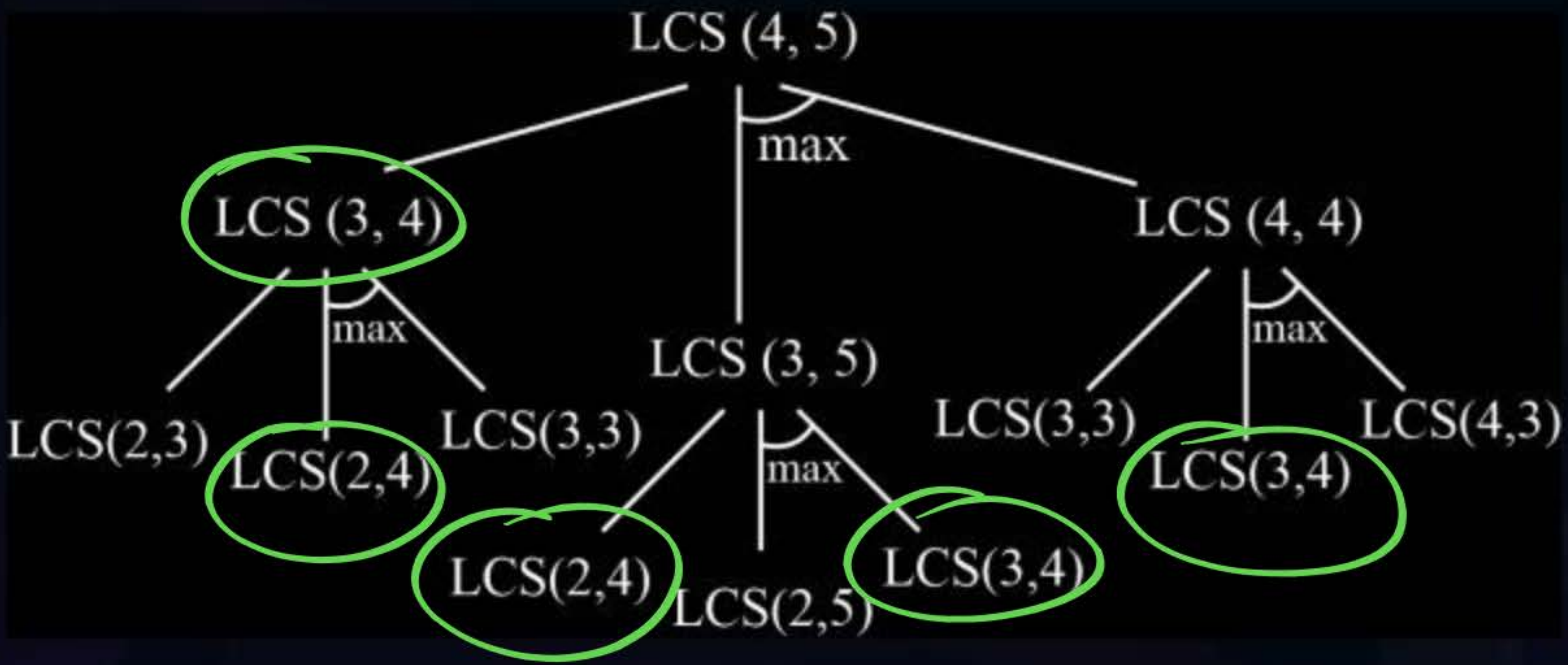
x = 4
y = 5



Topic : Dynamic Programming

Overlapping Subproblems:

$|X| = 4, |Y| = 5$



Overlapping exists

Topic : Dynamic Programming

Example:

Top-Down (Memorization/Recurrence):

X = "ABBAB"

Y = "ACBAB"

LCS : "ABAB"

Length = 4

ABBAB , ACBAB

1+

ABBA ACBA

1+

Example:

Bottom-up Approach (Tabulation):

X = "ABCBDAB" → length = 7

Y = "BDCABA" → length = 6

Topic : Dynamic Programming

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1	2	2	2	2
B	0	1	1	2	2	3	3
D	0	1	2	2	2	3	3
A	0	1	2	2	3	3	4
B	0	1	2	2	3	4	4

HW

Length of LCS

Topic : Dynamic Programming

BU

L[i–1, j–1]	L[i–1, j]
L[i, j–1]	$\boxed{\text{L[i, j]}}$

+1

Topic : Dynamic Programming

LCS Recurrence:

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

Topic : Dynamic Programming

Algorithm LCS based on Bottom-up Tabulation method :

Algorithm LCS (x, y)

Integer x[1..n], y [1..m];

```
{
        integer L [0..n , 0..m];
1.      for i ← 1 to n
                L [i, 0] = 0;
2.      for j ← 1 to m
                L [0 , j] = 0;
3.      for i ← 1 to n
                for j ← 1 to m
```

HW

Topic : Dynamic Programming

```
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]};
```

HW

THANK - YOU