

# DYNAMIC PROGRAMMING

## Fibonacci No.

$n = 0$	$1$	$2$	$3$	$4$	$5$	$6$	$\dots$
$0$	$1$	$1$	$2$	$3$	$5$	$8$	$\dots$

✓ In recursion →

psvm {

int  $n=5;$

cout (fib( $n$ ));

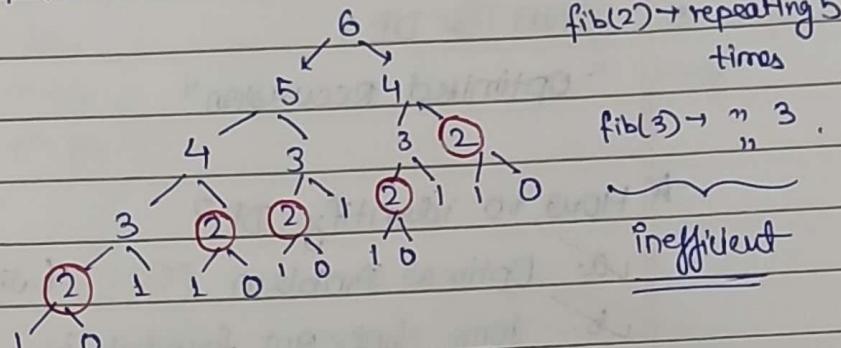
static int fib(int  $n$ ) {

if ( $n < 2$ ) {

return  $n;$

return fib( $n-1$ ) + fib( $n-2$ );

}



✓ fib(2) → repeating 5 times

fib(3) → " 3 .

inefficient

✓ How to solve this problem of inefficiency ?

⇒ ✓ Store it in an array i.e. save fib(2), fib(3) to values in array & store.

int f[] = new int[n+1]; → initially 0, 0, 0, 0 ..

↳ ↳ starting from 0<sup>th</sup> fibonacci to n<sup>th</sup> eg. n=6 ↓

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

✓ If fib( $n$ ) is already calculated

∴ (f[n] != 0) → return f[n]

✓ Storing continuously → storing these into f[n] ✓

$$f[n] = f[n-1, f] + f[n-2, f];$$

return f[n]

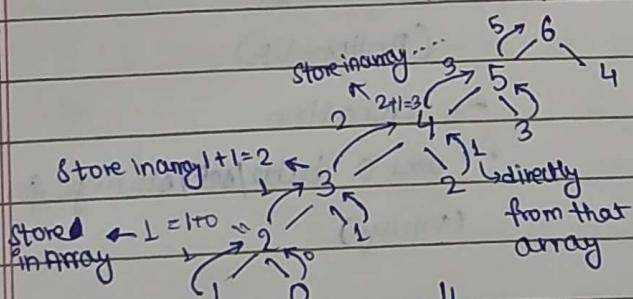
Hence,

earlier Time complexity was exponential but now

it's linear  $\rightarrow O(n)$

↓  
How?

$O(2^n)$



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

If we count operations here,  
for how many nos. fibo. was calc.

$0, 1, 2, 3, 4, 5, 6$

$m+1$

$\downarrow$   
 $O(n)$

∴ Skewed Tree  
(ext. array Jhukhehua  
rec. BT)

This whole process of storage of memory for optimization for other recursive calls (for reuse) is called Memoization.

## • What is DP?

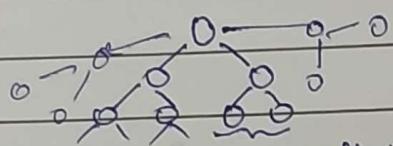
"Optimized Recursion"

### ★ How to identify DP?

a. Optimal Problem

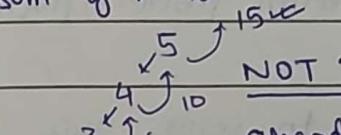
(like we did with Greedy)

b. Some choice got founded i.e. which forms multiple branches in rec. tree.

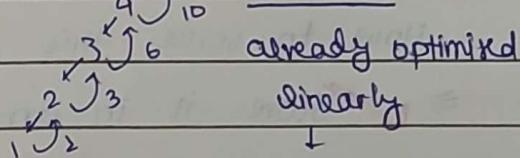


∴ we can eliminate  
this with help of  
thus,

e.g. sum of n nos.



NOT DP

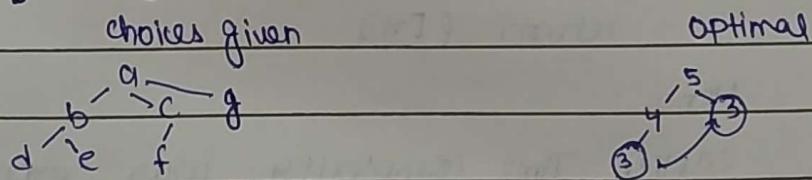


already optimized

linearly

better optimization =  $\frac{n(n+1)}{2}$ .

★ **Definition:** DP is a technique in programming that helps to efficiently solve a class of problem that have overlapping subproblems and optimal substructure property.



## Ways of DP

### ① Memoization

(Top Down)

e.g. Fibonacci

→ Recursion

→ Storage of subproblems  
for reuse.

### ② Tabulation

(Bottom Up)

→ Iteration

→ uses a table/set/storage for iteration  
(conchy)

Note:- Tabulation is more efficient in comparison to Memoization.

## ~~Tabulation Process~~

dp[0]	0	1				
	0	1	2	3	4	5

$\downarrow$   
 $dp[i] \rightarrow i^{\text{th}}$  fib.

$dp[0] \rightarrow 0^{\text{th}}$  fib

$dp[n] \rightarrow n^{\text{th}}$  fib.

$dp[n+1]$

for (int i=2; i<n; i++) {

$$dp[i] = dp[i-1] + dp[i-2]$$

ans. =  $dp[n]$

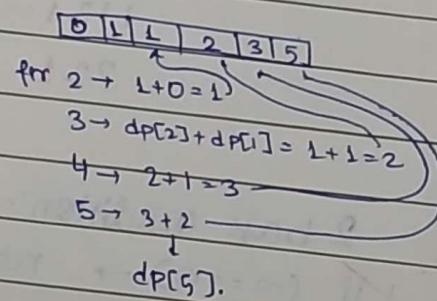
T.C here also  $\rightarrow O(L)$

★★★ steps 8-10  
already known values like 0, 1.

1) initialisation

2) meaning

3) filling: from small to large



5/08/24

## 7 important Questions/concepts

Fibonacci

Unbounded knapsack

0-1 knapsack

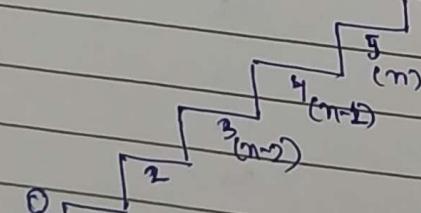
LCS

Kadane's Algo.

Catalan No.

DP on Grid (2D Arrays)

Q. Climbing Stairs  $\rightarrow$  [concept of Fibonacci no.]  
Count ways to reach the  $n^{\text{th}}$  stair. The person can climb either 1 stair or 2 stairs at a time.



ways for 3

111

12

21

ways 4

1111

121

211

112

22

Therefore, ways for 5 will be same as (ways of 4)+1  
 \* and (ways for 3)+2 [oooh! v.v. easy]

✓ ways for 5

1112      11111

122      1211

212      2111

1121

221

$n=5$

∴ 8 ways

113  
23

Hence,

$$\boxed{\text{ways}(n) = \text{way}(n-1) + \text{way}(n-2)}$$

~~1 step      2 step~~  
 (Don't need to add steps as we're just calc.)

2 base condition:-

✓ If ( $n=0$ ) → return 1; (Ground to Ground)      no. of ways not steps

✓ If ( $n<0$ ) → return 0;

∴ if  $n=1 \rightarrow \text{ways}(1) = \text{ways}(0) + \text{ways}(-1)$

~~out of bound if no condition~~

Now memoization :-  $O(n)$

(Analyze it by using skewed tree + storing in array)  
 (same as fibo. nos.)  $\Rightarrow$  check code!

Q. If asked we can climb 1, 2 & 3 stairs :-

ways for 2

ways for 3

ways for 4

11

111

1111

2

12

121

21

211

3

112

∴ ways of 5 = ways(2)+ways(2)+ways(4)

~~Step3      Step2      Step3~~

but ways will be same

22

13

31

Hence,

$$\text{ways}(n) = \text{way}(n-1) + \text{ways}(n-2) + \text{ways}(n-3)$$

My other problems.

## \* 0-1 Knapsack

- Types of knapsack problems
  - Fractional knapsack
  - 0-1 knapsack
  - Unbounded "
- e.g. of 0-1 knapsack Problem :-

	value	weight	
fridge →	100	10	
laptop →	50	5	$W = 10$
phone →	30	2	
tablet →	25	1	
pencil →	5	0.5	
book →	15	1	

✓ In this problem we select max. valued item for 10 kg knapsack.  
 [In case of 0-1 knapsack, apart from greedy (frac. knapsack) we don't take fractional values to achieve max. value for full 10kg, rather we here either we take it or ignore it for max. weight reached.]

case-1	case-2
fridge (full) $val = 100$	laptop, phone, tablet, pencil, book → 9.5 kg $value = 135$ ✓

e.g. for Unbounded Knapsack

case-1	case-2	case-3
"	"	10 tablets → $w_t = 10$ kg $value = 250$

Q.  $val[] = 15, 14, 10, 45, 30$

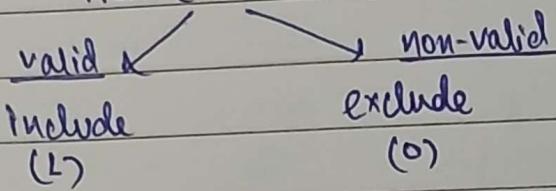
$wt[] = 2, 5, 1, 3, 4$ .

$W$  (total allowed weight) = 7  
max Profit?

→ How DP ques.?

- ⇒ 1. Choice (selecting numerous + putting in knapsack)
- 2. Optimal → when we need to find best sol from feasible sol possible.

Therefore :- item (val, wt)



$wt \leq w$

include      exclude  
(for selecting max.  
value)

base condition → Wcapacity , index at n to 0

at item index = 0

∴ if  $wt = 0$  so  
can be  
no items added so,  
ans = 0 .

∴ no item left now  
so ans = 0

Hence, → Knapsack (val[], wt[], w, n)

(recursion)

if ( $w = 0 \text{ || } n = -1$ )

return 0;

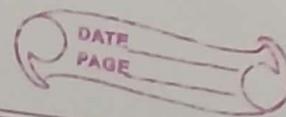
if ( $wt \leq w$ ) → valid

→ include  $\Rightarrow w - wt$ ,  $i-1$

→ exclude  $\Rightarrow w$ ,  $i-1$

else → not valid

→ exclude  $\Rightarrow w$ ,  $i-1$



### 0-1 Knapsack

[ ] val = 15, 14, 10, 45, 30

[ ] wt = 2, 5, 1, 3, 4

W = 7

Recursion tree →  $\sum_{i=1}^5 w_i = 45 + 30 = 75$

$$m = 5, W = 7$$

include

exclude

$$m = 4, W = 7$$

$$m = 4, W = 3$$

$$m = 3, W = 0$$

$$m = 3, W = 3$$

not valid

$$m = 2, W = 2$$

$$m = 2, W = 3$$

exclude

$$m = 1, W = 2$$

$$m = 0, W = 0$$

$$m = 0, W = 2$$

$$m = 0, W = 1$$

$$m = 0, W = 3$$

$$m = 3, W = 4$$

$$m = 2, W = 3$$

$$m = 2, W = 4$$

$$m = 1, W = 3$$

$$m = 1, W = 4$$

$$m = 0, W = 2$$

$$m = 0, W = 3$$

$$m = 0, W = 4$$

$$m = 1, W = 1$$

$$m = 0, W = 1$$

$$m = 0, W = 2$$

$$m = 0, W = 3$$

$$m = 0, W = 4$$

$$m = 0, W = 5$$

$$m = 0, W = 6$$

$$m = 0, W = 7$$

$$m = 0, W = 8$$

$$m = 0, W = 9$$

$$m = 0, W = 10$$

$$m = 0, W = 11$$

$$m = 0, W = 12$$

$$m = 0, W = 13$$

$$m = 0, W = 14$$

$$m = 0, W = 15$$

$$m = 0, W = 16$$

$$m = 0, W = 17$$

$$m = 0, W = 18$$

$$m = 0, W = 19$$

$$m = 0, W = 20$$

$$m = 0, W = 21$$

$$m = 0, W = 22$$

$$m = 0, W = 23$$

$$m = 0, W = 24$$

$$m = 0, W = 25$$

base condition: if ( $m = 0 \text{ || } W = 0$ )  
return 0;

In case of valid use Math.max for include & exclude at every step return call.

✓ Memoization → val[], wt[] → fixed ✓

w, m → variables in recursion → 2D array ✓

array { from 0 to n+1 items } ↗ val.length

↓ and 0 to w+1 weights

int[] dp = new int[n+1][w+1]

initialize all with -1 ✓

& if ( $dp[m][w] \neq -1$ ) → filled/already stored

∴ return  $dp[m][w]$

in matrix  $dp[i][j]$

Knapsack(w) = j

items = 0 to i

for last row, last col → i = n

j = w

→ check code

→ Time Complexity :  $O(n \times w)$  ~ size of array

• Tabulation → Create a Table : → base cond = 0, 1, 2, 3, 4, 5, 6 ↗ max. wt.

$(n+1) \times (w+1) \rightarrow$

DP[][](i,j) → i = items + j = w (knapsack) size

	0	1	2	3	4	5	6
0	✓						
1		✓					
2			✓				
3				✓			
4					✓		
5						✓	
6							✓

✓ Meaning & initialize

at index i,j let (2,3) → m=2, w=3 → smaller problem of (1,5,2) + (1,4,5).

work. no. (n+1) → size array hoga tabhi

m → achieve hoga★★★

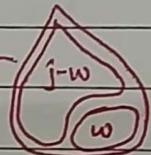
initialize → (Base case) →  $W=0 \therefore \text{profit} = 0$  ✓  
 $\boxed{n=0 \therefore \text{"no item"}}$  ✓

### (5) Filling (bottom up)

all 0th row + 0th row (0) → 0 for using Base condn  
 now → for (int  $i=1$  to  $n+1$ ) {  
 allowed weight (capacity) ← for (int  $j=1$  to  $W+1$ ) {  
 $\boxed{\text{val}[i-1], \text{wt}[i-1]}$  } } is me galti kabhi mt karna!  
 if ( $\text{wt}[i-1] \leq j$ ) // valid, since  $j \rightarrow \text{capacity at that}$   
 now for this inside array early 2nd index is now at cell or, i  
 1st index. at  $(i,j)$  weight capacity is  $j$  ✓

I. Valid Case 1 : include

value of what item ← val[i-1] + dp[i-1][j-wt] ← now to calc max. profit of  
 15 14 10 45 30 0 items ✓ e.g. if (3,3) →  $i=3, W=3 \text{ kg}$   
 2 5 1 3 4 has  $\boxed{I_1(15,2 \text{ kg}), I_2(14,5), I_3(10,1)}$



①  $I_3 \rightarrow$  Selected  $\text{wt.} = 1 \text{ kg} \therefore \text{left } 3-1=2 \text{ kg}$  [j-wt] ✓

② Now check for filling in  $I_1 + I_2$  ✓  $\boxed{dp[i-1]}$  ✓

Case 2 : exclude

$dp[i-1][j]$  ✓ (simple)

$\therefore dp[i][j] = \max(\text{case 1, case 2}) \rightarrow \text{store in } dp.$  ✓

II. Invalid  $\rightarrow dp[i-1][j] = dp[i-1][j]$

final ans return  $\rightarrow dp[n][W]$  which means all items considered under

✓ since last wala hi saara capacity. ✓  
 can consider 1carrage

$dp[i][j] \rightarrow dp[i-1][j] \Rightarrow \text{for this } W=1 \text{ kg}$

→ check dp array  
 n = 1 item only.

For better understanding watch how dp array filled in video.

## \* Target Sum Subset

Variation of 0-1 Knapsack

numbers[] = 4, 2, 7, 1, 3

Target sum = 10  $\rightarrow$  True (can be achievable)

How 0/1 Knapsack

$\therefore$  Subsets  $\Rightarrow \{7, 3\}, \{7, 2, 1\}, \{4, 2, 3, 1\}, \dots$

How DP?  $\rightarrow$  (1). Choice of elements (p, up)

(2). limit on max. allowed capacity

(3). val = wt

$$\text{var} = \text{wt}$$

target sum  $\rightarrow$  max. allowed capacity

Using Tabulation:- i  $\underbrace{j}_{\text{(target sum)}}$

1. Table.  $\rightarrow dp[n+1][sum+1]$

Unlike in 0/1 knapsack we were

storing quantities as max Profit but

here we have to check True/False

2. meaning + initialize

3. bottom up manner (filling)  $\rightarrow$  small to large.

\* meaning  $\rightarrow$  (boolean type T/F)

ans  $\rightarrow$ , n items  $\Rightarrow$  subset sum = target? T/F

(smaller problem) \*  $dp(i, j) \rightarrow$  i item  $\Rightarrow$  subset sum = j? T/F

e.g.  $dp(3, 5) \rightarrow$  3 items subset sum = 5 (j)

using [4, 2, 7] we can't make 5  $\therefore$  False.

{}, {4}, {4, 2}, {7}, ...

initialise (base case)  $\rightarrow$ , sum = 0  $\therefore$  True using {} /  $\emptyset$ .

(j=0)

2. items = 0 ( $i=0$ )  $\rightarrow$  sum > 0  $\therefore$  False always  
(j=1 to sum)

$\therefore$  0<sup>th</sup> row from index 1  $\rightarrow$  all False

0<sup>th</sup> col  $\rightarrow$  all True

Note:- in boolean type method, ~~array~~ array is initialised using  
False only.  $\therefore$  No need to declare this.

Filling  $\rightarrow$  small  $\rightarrow$  large

i↓ j↓      i↑ j↑

30

~~check~~  $\rightarrow$  Note: val[i-1] was used becoz at base cond<sup>n</sup> row + col theres an extra (0) -

## \* Rod Cutting -

Given a rod of length  $n$  inches and an array of prices that includes prices of all pieces of size smaller than  $n$ . Determine the max. value obtainable by cutting up the rod and selling the pieces.

Length = 1 2 3 4 5 6 7 8

price = 1 5 8 9 10 17 17 20

Rod length = 8 .

$$\text{case 1} \rightarrow 4, 4 \\ \downarrow \quad \downarrow \\ 8 + 8 = 16$$

$$\text{case 2} \rightarrow 4, 2 \\ \downarrow \quad \downarrow \\ 17 + 1 = 18$$

$$\text{case 3} \rightarrow 2, 6 \\ \downarrow \quad \downarrow \\ 5 + 17 = 22$$

: same as knapsack problem. "for Max. Profit".

"Unbounded knapsack"

price = val

length = wt

total Rod = w

code :-

```
fun ( int i=1, i<n+1, i++ ) {
```

```
    " " j " " w<wt[i] " " }
```

if ( wt[i-1] <= j ) { // valid

dp[i][j] = Math.max(val[i-1]+dp[i][j-wt[i-1]],

else { // invalid . dp[i-1][j]);

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

Here,  $dp(i, j)$  = for 'j' total rod length how many pieces do we need to get for max. Profit . . .

Y

else { // invalid .

dp[i-1][j]);

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

}

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

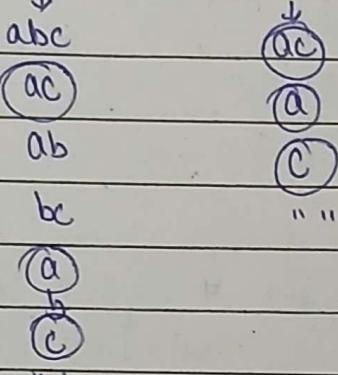
★ \*  
IMP:

## Longest Common Subsequence (LCS)



A Subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of remaining characters.

e.g. abc and ac



∴ from both common subsequences  
one with longest (length) size  
is ac  
∴  $ans = 2$ .

e.g. abcde and ace  $\Rightarrow ans = 3$

recursion approach :- (1) Base condition

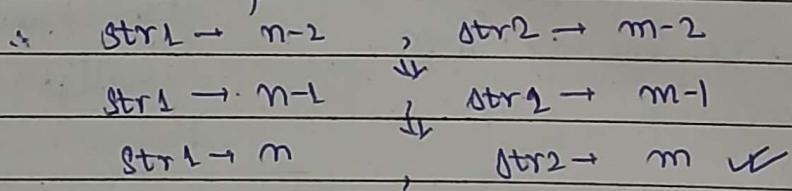
(2) Bigger to smaller problem.

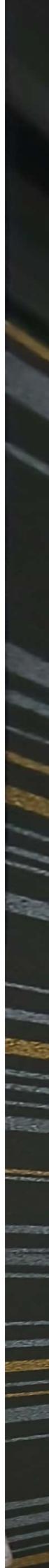
$str1 = "abcde"$ ,  $str2 = "ace"$

$ans = 3$  "ace"

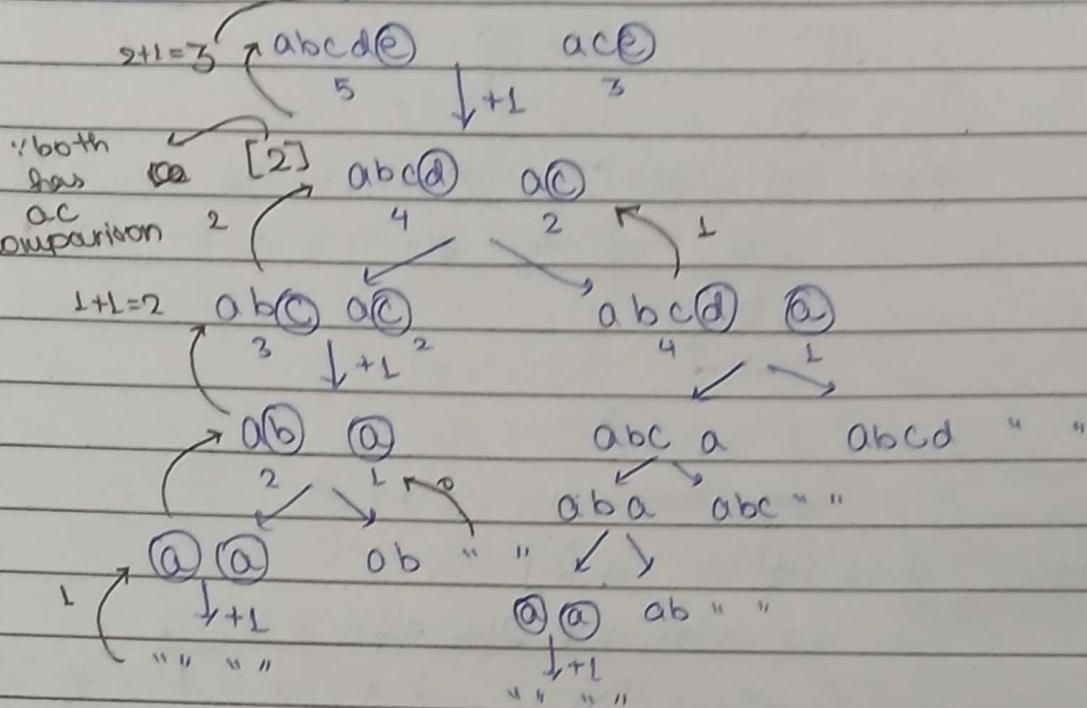
for tot checking in strings of sizes m + n first

find for smaller problem.





Recursive tree for 3rd Ques.



1. Concept "code"  $\Rightarrow \text{JCS}(\text{str1}, \text{str2}, n, m)$

Base case  $\rightarrow \text{if } (n == 0 \text{ || } m == 0)$

return 0;

Same  $\rightarrow \text{if } (\text{str1}(n-1) == \text{str2}(m-1))$

return  $\text{JCS}(\text{str1}, \text{str2}, n-1, m-1) + 1$

Different  $\rightarrow \text{else ans1} = \text{JCS}(\text{str1}, \text{str2}, n-1, m)$

$\text{ans2} = \text{JCS}(\text{str1}, \text{str2}, n, m-1)$

} return  $\text{Math.max}(\text{ans1}, \text{ans2})$

## • Memorization approach



Because of overlapping subproblems

∴ We make a 2D array since 2 variables are changing (n and m) after every recursion call.

for  $\rightarrow$        $str1 = "ABCD"$        $str2 = "ACEB"$   
 $m=4$                    $m=4$

1.  $n+1 \rightarrow$  rows  
 $m+1 \rightarrow$  cols.  
 $\downarrow$   
 $dp[n+1][m+1]$

meaning  $\rightarrow$  at  $(i,j)$  which string will be stored:  
 $\rightarrow$  e.g. abc ( $m=3$ ), ccc ( $m=2$ )  $\therefore$  at  $(3,2)$   
e.g. ABC ( $m=3$ ); ACE ( $m=3$ )  $\therefore$  at  $(3,3)$ .

2.  $(i,j) \rightarrow str1(i), str2(j)$   
 $\downarrow$   
LCS

for final ans  $\&$   $i=m + j=m$  (last row, last col).

Hence, fill all rows + col with -1.

+ if  $(dp[n][m] \neq -1)$   
return  $dp[n][m]$   
else  $\rightarrow$  LCS  $\rightarrow$  same / diff.

Time complexity:  $O(n \cdot m)$ .

### Tabulation approach

$\rightarrow$  helps in preventing from recursion, because of stack overflow issue.

3 Steps:- ①. table create  $\rightarrow dp[n+1][m+1]$

②. meaning + initializing

③. fill (bottom up)      (base condition)

$\downarrow$   
from small to large  $\rightarrow$  value for smaller problem  $\therefore$  eventually can value for larger problem.

for eg.  $\rightarrow$        $str1 = "abcde"$        $str2 = "ace"$

$str1(n)$ ,  $str2(m)$   
 $\downarrow$        $\downarrow$   
 $i$        $j$

meaning  $\rightarrow$  for  $(i,j) \rightarrow str1(i) \rightarrow$  LCS?  
 $str2(j)$

e.g.  $(3,2) \rightarrow i=3$  "abc"  $\rightarrow$  LCS? = ac  $\therefore @ (3,2)$   
 $j=2$  "ac"      "ac"  
in stored

initialise → base condition → same as of recursion → that is  
length of str<sup>1</sup> or 2 is zero ∴ return 0.

@ i=0 and j=0

$\therefore$  1st row + 1st col will be zero.

Comparing last character of both strings

NOW In rec<sup>k</sup>  $\rightarrow$   $\overbrace{\text{same}}^{\therefore +L}$   
 here  $\rightarrow$   $\underbrace{i-1 \rightarrow j-1}_{\text{same}} + L$

$$\therefore dp[i][j] = dp[i-1][j-1] + L.$$

else

Ans1 = dp[i-1][j]

$$ans_2 = dp[i][j-1]$$

$dp[i][j] = \max(\text{ans}_1, \text{ans}_2)$ .

• Time Complexity:  $O(n^m)$

## \* Longest Common Substring

Note:- a substring is a contiguous sequence of characters within a string.

e.g.  $S_1 = "ABCDE"$ ,  $S_2 = "ABACE"$

longest common subsequence = ABCE  
 " " substring = AB

\* e.g. 2  $S_1 = "ABDEFG"$ ,  $S_2 = "ABCDEFAM"$

L.C. substring = DEFA

### • Tabulation approach :-

① 2D array ( $m \times m$ )

$\Rightarrow$  characters same  $\rightarrow$  length + 1

② meaning, initialise

$\Rightarrow$  " different  $\rightarrow 0$

③ filling up.

$str1, m$      $str2, m$     substring  
 $\downarrow$   
 $str1, m-1$      $str2, m-1$     substring

1. Now  $\rightarrow$  rows :-  $0 - m+1$  } +1 since calculating from 0 to n  
 cols :-  $0 - m+1$  } 0 to m

2. meaning  $\rightarrow (i, j) \rightarrow str1(i), str2(j)$

e.g.  $i=1$  (A)

longest common

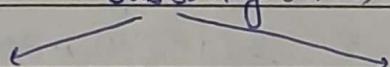
$j=2$  (AB)

substring will be stored

$\therefore$  at  $(1, 2) \rightarrow A \rightarrow 1$

3. initialisation  $\rightarrow n=0, m=0 \rightarrow \therefore 0$ .

substring ( $n, m$ )



same

$\therefore (m-1), (m-1)) + 1$

$dp[i][j] \uparrow$

different

$dp[i][j] \rightarrow 0$

counter resets

Year

Note:- here it's not certain that the final answer will be at  $(i=n, j=m)$ , here it can be present even at any cell not considering full string length.

$$\therefore \text{ans} = 0$$

$$\text{ans} = \max(\text{ans}, \text{dp}[i][j]).$$

Time complexity:  $O(n^2 m)$ .

20/08/24

\* Longest increasing Subsequence

$$\Downarrow \text{arr}[] = \{50, 3, 10, 7, 40, 80\}$$

Length of LIS = 4.

meaning longest ascending (sorted)  $\rightarrow [3, 10, 40, 80] \rightarrow 4$   
 $\downarrow$   
 $4, 40, 80$       longest

$$\text{eg. arr}[] = \{5, 1, 6\},$$

$$\text{LIS} = 2 \rightarrow [5, 1, 6, 5, 1, 6, 5, 1, 6]$$

all sub-  
sequences

longest sorted increasing  
subsequence

Solving approach: eg.  $\{50, 3, 10, 7, 40, 80\} - A$

sorted  $\Rightarrow \{3, 7, 10, 40, 50, 80\} - B$

If we'll find the LCS of A and B we'll get:-

$$\Rightarrow [3, 10, 40, 80] \rightarrow \text{i.e. } 4 \Rightarrow \underline{\text{ans}}$$

Hence, LIS = Longest common sorted unique subsequence.

and if we've got any duplicate elements then take only 1.

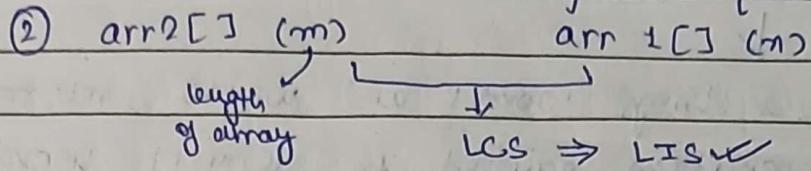
e.g.  $50, 50 \equiv 50$

Teacher's Signature.....

"Benefit of HashSet is that it stores only unique elements."

Date		
Page No.		

∴ Steps :- ① HashSet → used to find unique elements.



→ m = n when elements were already unique.

Time Complexity:  $O(n^2m)$ .

Code ⇒ public static int lcs (int [] arr1) {

    → HashSet < Integer > set = new HashSet <> ();

    for (int i = 0 to arr1.length) {

        set.add(arr1[i]);

    int [] arr2 = new int [set.size()]; → sorted unique  
    elements wala  
    array.

    int r = 0;

for all numbers in → for (int num : set) {

the set

    arr2[r] = num;

→ storing those nos. in arr2

    r++;

}

Arrays.sort(arr2); → ascending sorted

return lcs (arr1, arr2);

lcs wala code ⇒ static int lcs (int [] arr1, int [] arr2) {

(for arrays)



concept saurjo + concept revise

hare raho tabhi yaad nahega.

Koi ek sheet select karo and  
use lagao.

\* \* \* Imp. for placements :- ① DSA ki sheets (Ques.) ex. 450 que.

② Web dev karo + projects banao

③ OS + DBMS

20|08|24

## \* Edit Distance

1

- Given 2 strings word1 and word2, return the min. no. of operations required to convert word1 to word2.

Using :-

  - (1). Insert a character
  - (2). Delete " "
  - (3). Replace " "

e.g. word 1 = "intention", word 2 = "execution,

intention → ~~t~~<sup>remove 't'</sup> → intention  
① ② | replace 'i' with 'e'

Exention ③ replace enention

④ replace 'in' with 'u'  
exclusion  $\xrightarrow[\text{u}]{\text{insert}}$  execution

dim. = 5

Similarity from LCS :-

word1: abcdef (n)

Word 2: bdeg (m)

①. same if last character

if both words are same.

② ~~the~~ Different

(a). Add

Suppose we added 'g' in word L.

∴ abcdef (g)

b d e g

$$\int \text{str1}(m) \text{ str2}(m-1) + 1$$

Since, now comparing b/w  
 $m \rightarrow abcdef$  and  $(m-1) \rightarrow bde$

(b) Delete

word1: abcde~~x~~ ] ∵ now compare in :-  
word2: bdeg

v. easy (ichud se socho why)

str1(m-1) str2(m) + 1

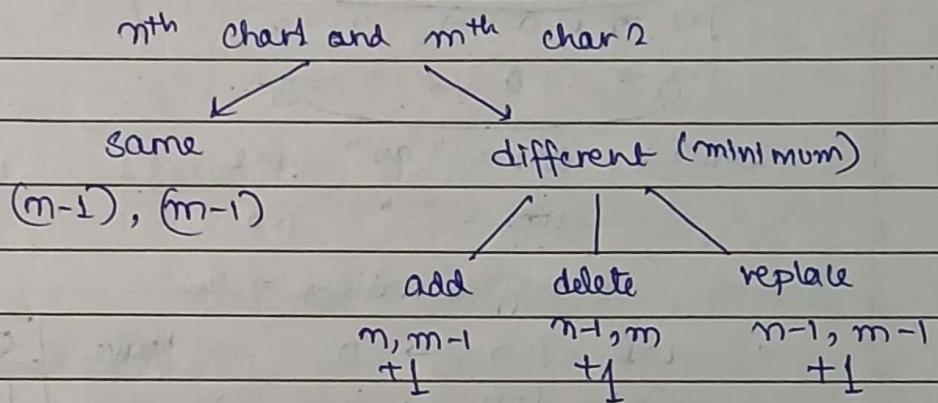
(c) Replace

1: abcde~~f~~ → replace with 'g' Now compare in :-  
2: bdeg

str1(m-1) str2(m-1) + 1

Ans. for different will be minimum of (a), (b), (c).

Tree:-



Hence, similarity:-

LCS

2 strings

Output: int (length of lcs)

Edit Distance

2 strings

Output: int (no. of operations)

∴ Tabulation approach:-

(1). 2D array : dp[n+1][m+1]

(2) meaning : e.g. "abc" "gbd"  
n=3 m=3

dimag ka  
we karo

in 2D array e.g. (1,2) → i=1, j=2

at (1,2) ⇒ 2 will be stored

a  
replace with 'b'

S → add 'b' → Sb ≈ 2 operations

initialisation: if one of string 1 is empty "" then the no. of operations will be just the length of str2 (all the elements hai in str2 usko ek-ek karke add kardo in str1 or vice versa → remove)

e.g.

i.e. case 1: str1 = " " i=0

str2 = "abd" j=3

∴ no. of operations = 3.

if j=0 → i=0 → no. of operation = 0

$\begin{matrix} j=1 \\ \downarrow \end{matrix} \rightarrow i=0 \rightarrow " = 1$

$\begin{matrix} j=2 \\ \downarrow \end{matrix} \rightarrow i=0 \rightarrow " = 2$

$\begin{matrix} j=3 \\ \downarrow \end{matrix} \rightarrow i=0 \rightarrow " = 3$

case 2: str1 = "ab" i=2

str2 = " " j=0

∴ no. of operations → 2.

∴ i=0 j=0 → 0

i=1 j=0 → 1

i=2 j=0 → 2

i=3 j=0 → 3

Hence,

0	1	2	3
1			
2			
3			

pseudocode for initialisation: for (int i=0 to n+1)

for (int j=0 to m+1)

if (i==0) → dp[i][j] = j

if (j==0) → dp[i][j] = i

(B) smaller to large "filling":-

here i = length of string 1

j = " " " " 2

Same ⇒ if (str1(i-1) == str2(j-1)) {

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

else

Diff. ⇒  $dp[i][j] = \min(\underbrace{dp[i][j-1]}_{\text{add}}, \underbrace{dp[i-1][j]}_{\text{delete}}, \underbrace{dp[i-1][j-1]}_{\text{replace}}) + 1$

Teacher's Signature.....  
Date \_\_\_\_\_

Date		
Page No.		

Note (Reminder) :- We can't compare 3 numbers in min operation, therefore we it nested wise.

Time Complexity:  $O(n \times m)$ .

→ code yourself → v.v.v. easily.

### • (Variation) String Conversion

Convert string<sub>1</sub> to string<sub>2</sub> with only insertion & deletion.

"Not Directly"  $\Delta$  → (normally gives ans. = 4)  
but using this approach  
ans. = 3.

e.g. str<sub>1</sub> = "pear" str<sub>2</sub> = "sea"

remove p, remove r, add s →  $\underbrace{\text{sea}}_{\text{LCS}}$

Note:- If we want to convert str<sub>1</sub> to str<sub>2</sub> :- Indirect (quick) method

- Find LCS (of both) and delete extra characters of string<sub>1</sub>.

str<sub>1</sub> = abcdef  $\rightarrow$  LCS = ace  $\therefore$  remove b, d, f from str<sub>1</sub>.

str<sub>2</sub> = aceg  $\downarrow$  now add 'g'

- Now add remaining.

1. No. of delete operations  $\rightarrow$  length of str<sub>1</sub> - length of ~~LCS~~ <sup>ace</sup>  

$$6 - 3 = 3$$

2. No. of insert operations  $\rightarrow$  length of str<sub>2</sub> - length of <sup>aceg</sup> ~~LCS~~  

$$4 - 3 = 1$$

$$\therefore 3+1 = 4 \text{ ans.}$$

Code yourself it easy.

21/08/24

## Wildcard Matching



Given a text and a Wildcard Pattern, implement wildcard pattern matching algorithm that finds if wildcard is matched with text. (it must cover entire text not partial).  
Wildcards include '?' and '\*'.

'?' - matches any single character. (but not with  $\emptyset$ )

e.g. text = "ab"       $P = "a?"$

replace ? with b

∴ "ab" ✓

'\*' - matches any sequence of characters (including empty sequence)

e.g. text = "aa" ,  $P = " * "$

↓  
replace aa with \*

If matching possible = True

not possible = False.

e.g.1 Text = "baaababab"

Pattern = " \* \* \* \* ba \* \* \* ab"

Output = true . replace aab with  $\emptyset$

e.g. Text = "baaababab"

Pattern = "a\*ab"

Op = false.

Variation of  
LCS

①  $n, m = \text{length of both strings}$

②  $m \downarrow, m \downarrow \rightarrow \text{smaller problems} \rightarrow s = "abc\underset{n}{e}f" p = "ab\underset{m}{?}d"$

1 Create of table of 2D array.  
 $\therefore n, m$

$(n-1) s = "abcde"$

$p = "ab\underset{m}{?}d"$

$(m-1)$   $s = " "$   $p = "ab\underset{m}{?}d"$

$(m=0) s = " "$

$p = " "$  ( $m=0$ )

2. assign meaning + initialise

$dp(i, j) \rightarrow$   $s(i) > T/F$  ~~with~~  $\Rightarrow$  check at  $(i, j)$  that it's possible or not.

e.g.  $i=1, j=4 \Rightarrow s = "a", p = " * ? b"$

~~not possible~~

$\therefore$  at  $(1, 4) \rightarrow \text{False will be stored.}$

initialise — (1).  $s=0, p=0$

Now since length of  $s \rightarrow i$

and " "  $p \rightarrow j$

$\therefore$  at  $i=0$  and  $j=0$

and  $s = " ", p = " " \rightarrow \text{True} \therefore \text{at } (0, 0) \rightarrow \text{True}$

(2). for  $s \neq 0, p=0 \nrightarrow \text{khali string pattern kabhi kisi string de match nhi kar skti (whose length} \neq 0)$

e.g.  $s = "aa", p = " " \rightarrow \text{False. (can't make aa from empty string)}$

$\therefore$  for  $j=0, i \neq 0 \rightarrow \text{False}$

(3) for  $s=0, p \neq 0$

$\begin{cases} s = " " & p = "?" \rightarrow \text{True}, P = "a\underset{?}{b}\underset{?}{c}" \rightarrow \text{False} \\ s = " " & p = "?" \rightarrow \text{False} \\ s = " " & p = "b" \rightarrow \text{False} \end{cases}$

but what about these

Ans, S=" " p = " \* a ? "

$\downarrow$  character  
Position  
 $\downarrow$  False

∴ state for  $dp[0][j-1]$

e.g. S=" " , p = " \* ? "  $\rightarrow$  True

S=" " , p = " a ? "  $\rightarrow$  False.

pseudo code  $\rightarrow$  for (int j = 1 to n)

if (p.charAt(j-1) == '\*' )

$dp[0][j] = dp[0][j-1]$

test }

Take, \* mils in 'p' check for last filled  
block jo usko hoga wohi arms hoga.

	0	1	2	3	4	5	
0	T	T	T	F	F	F	$s = "abc" \rightarrow n=3$
1	F						$p = " * ? b c" \rightarrow j=5$
2	F						(1). $0,1 \rightarrow s = " ", p = " * "$
3	F						(2). $(0,2) \rightarrow s = " " p = " * ? "$ start dash $\downarrow$ dash per. for prev. block
							(3). $(0,3) \rightarrow s = " " p = " * ? b ? "$ $\downarrow$ at (0,3) T (after)
							? : false      True, True
							(4). $(0,4) \rightarrow p = " * ? b ? " \text{ False}$
							(5). $(0,5) \rightarrow p = " * ? b ? "$
							last j < * , block previous

### 3. Filling (bottom up)

Q: If  $s = a$  e.g.  $s = "abc" , p = "adc"$   
 $\downarrow$   $\downarrow$   $\rightarrow$  same  $\leftarrow$   $\downarrow$

filling in array  $\leftarrow dp[i][j] = dp[i-1][j-1]$

b. e.g.  $s = "abab" , p = "ab?"$

? can be replaced  $\rightarrow ? = ?$   
with & char.

Fals  $\leftarrow$  False

$\therefore \text{if } (\text{s.charAt}(i-1) == \text{p.charAt}(j-1)) \text{ || } \text{p.charAt}(j-1) == '?' \text{ } \}$   
 $\quad \quad \quad \text{dp}[i][j] = \text{dp}[i-1][j-1]$

1<sup>st</sup> Case  $j \rightarrow s = "abcd"$        $p = "aab"$       case  
 ignore       $s=i-1, p=j$   
 $\text{dp} = " "$       + compared now  
 $\therefore s=i, p=j-1$       since it can take multiple

Items - It will check also for  
 if any of them      next item.  
 gives true : True - we 'OR'

Therefore  $\rightarrow$  if ( $\text{p.charAt}(j-1) == '?'$ ) {

$$\text{dp}[i][j] = \text{dp}[i][j-1] \text{ || } \text{dp}[i-1][j].$$

2<sup>nd</sup> Case  $\text{dp}[i][j] == \text{false}$ .

Time Complexity =  $O(n^m)$ .

### \* Catalan's Number ★★★

$$\text{Fixed} \Rightarrow [C_0 = 1, C_1 = 1]$$

{ same as fibo. }

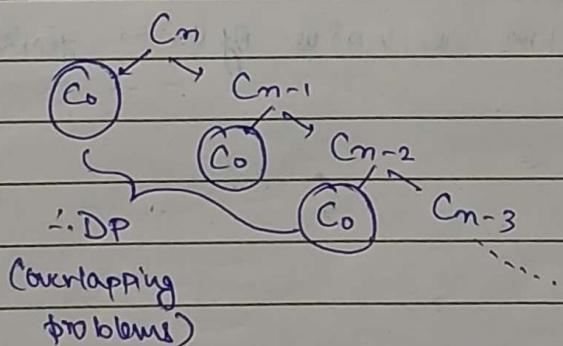
$$\therefore C_2 = C_0 \cdot C_2 + C_1 \cdot C_1 + C_2 \cdot C_0 = 5$$

$$C_2 = C_0 C_1 + C_1 C_0 = 2$$

$$\therefore [C_n = C_{n-1} \cdot C_0 + C_{n-2} \cdot C_1 + C_{n-3} \cdot C_2 + \dots + C_0 \cdot C_{n-1}]$$

$$\text{e.g. } C_4 = C_3 C_0 + C_2 C_1 + C_1 C_2 + C_0 C_3 = 5 + 2 + 2 + 5 = 14.$$

Why DP:-



## I. Recursion Approach

∴ for 0 we have  $n-1 \geq i=0 \therefore n-0-1 = n-1$

1 →  $n-2 \geq i=1 \therefore n-1-1 = n-2$

2 →  $n-3 \geq i=2 \therefore n-2-1 = n-3$

$[i \rightarrow n-i-1] \leftarrow$

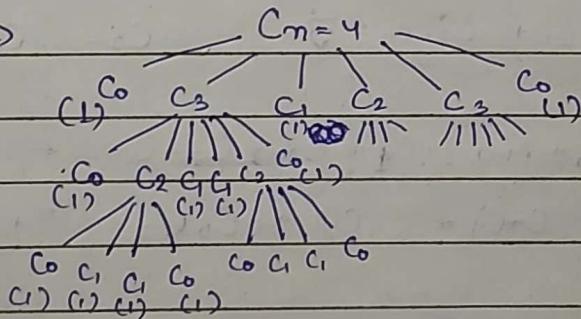
\* for (int i=0 to n-1)

$C_n += \underbrace{C_{n-i-1} + C_i}$

e.g.  $C_3 = C_2 C_0 + C_1 C_1 + C_0 C_2 \leftarrow$

✓ Pseudo Code \*  $\Rightarrow$  catalan(n) {  
 if ( $n=0$  ||  $n=1$ ) {  
 return 1  
 }  
 int ans = 0;  
 for (int i=0 to n-1) {  
 ans += catalan(i) \* catalan(n-i-1)  
 }  
 return ans.

• rec. Tree  $\Rightarrow$



here if you will store the value of  $C_2 \rightarrow$  you'll save 16 such calls!

## memoization Approach

(1D array) → ~ "n"

Fill dp array with -1. of  $0 \rightarrow 4 \therefore (0+n+1)$ .

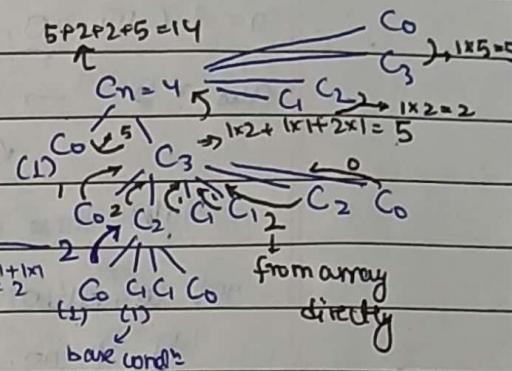
$\Rightarrow \text{cat}(n) \}$

if ( $n = 0$  ||  $n = -1$ )

return 1;

```
if( dp[m] != -L )  
    return dp[m]
```

```
for (int i=0 to n-1) {
    if (arr[i] < arr[i+1])
        swap(arr[i], arr[i+1]);
}
```



$$\therefore \text{dp } \begin{bmatrix} 1 & 1 & 2 & -1 & -1 \\ 0 & 1 & 2 & 3 & 4 \end{bmatrix}$$

## Tabulation Approach

✓. table

→ 1D array  $dp[m+1]$

✓. initialize

$$d\rho[0] = d\rho[1] = 1$$

### 3. filling

for(int i=2 to n) { } for filling in dp array

```
for (int j=0 to (-1))
```

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

for calculating  ~~$\alpha$~~   $C_i$

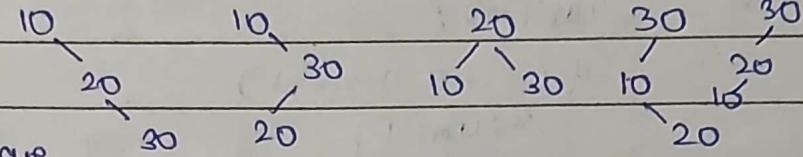
dry run  
& check for  
yourself.

## \* Counting Trees [type of Catalan No. Ques.] → think wisely

Find no. of all possible BSTs with given  $n$  nodes.

$$n=3 \quad (10, 20, 30)$$

$$\text{Ans} = 5 \rightarrow$$



Note:- these nos. are

not compulsory, agar koi  
bhi 3 no. le le tab bhi

5 hi aayega.

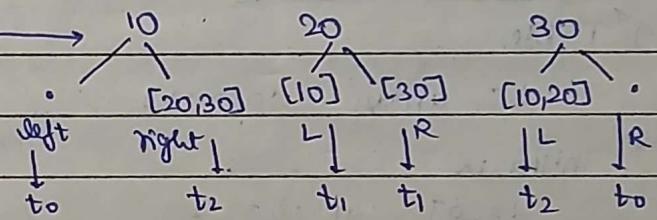
BST

$$n=0 \quad 1 \quad (\text{null node})$$

$$n=1 \quad [10] \quad 1 \quad (10)$$

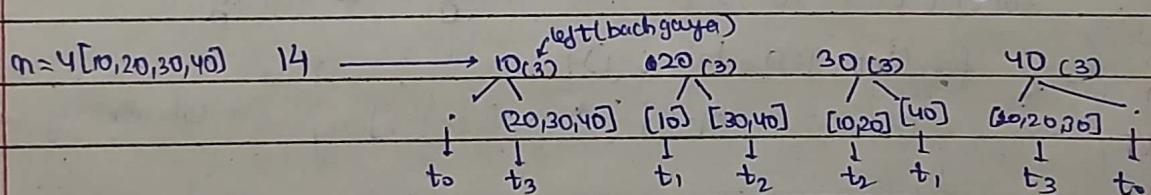
$$n=2 \quad [10, 20] \quad 2 \quad \begin{matrix} 10 \\ 20 \end{matrix} \quad \begin{matrix} 20 \\ 10 \end{matrix}$$

$$n=3 \quad [10, 20, 30] \quad 5$$



$$\Rightarrow t_0 \cdot t_2 + t_1 \cdot t_1 + t_2 \cdot t_0$$

$$= 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 5.$$



$$\Rightarrow 2 \cdot 5 + 1 \cdot 2 + 2 \cdot 1 + 5 \cdot 1 = 14.$$

expand the tree & verify.

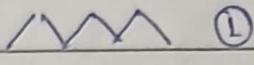
Hence,

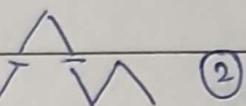
e.g.  $n=5 \Rightarrow$  Catalan of 5  $\Rightarrow$  No. of BSTs.

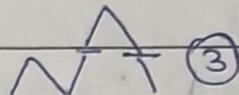
### • Mountain Ranges

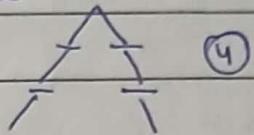
at any moment the no. of down strokes cannot be more than no. of up strokes.

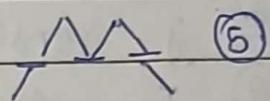
$\backslash$  = up stroke       $/$  = down stroke

pairs = 3  $\Rightarrow$   ①

5 combinations  ②

 ③

 ④

 ⑤

Hence,

$\therefore$  Pairs = n  $\Rightarrow$  Catalan of n  $\Rightarrow$  Catalan of n Combinations.  
Ans

### \* Matrix Chain Multiplication $\rightarrow$ "DP on grid"

(7 concepts in DP)

$$\text{arr[]} = \{1, 2, 3, 4, 3\}$$

Find min. cost.

matrix:      A      B      C      D  
order:      1x2      2x3      3x4      4x3

You firstly have to check the multiplication of ABCD is feasible or not then look for min. cost [cost  $\Rightarrow$  for  $A_{1x2} B_{2x3} \Rightarrow 1x2x3$  ]

e.g.      A      B      C  
              1x3      3x5      5x6

$\checkmark ((AB)C)$        $((A(B)C))$

AB: result = 1x5, cost = 1x3x5 = 15 : A1

A1.C: result = 1x6, cost = 1x5x6 = 30 : A2

Total cost = 30 + 15 = 45

BC: result = 3x6, cost = 3x5x6 = 90 : A1

(feasible multiplication)

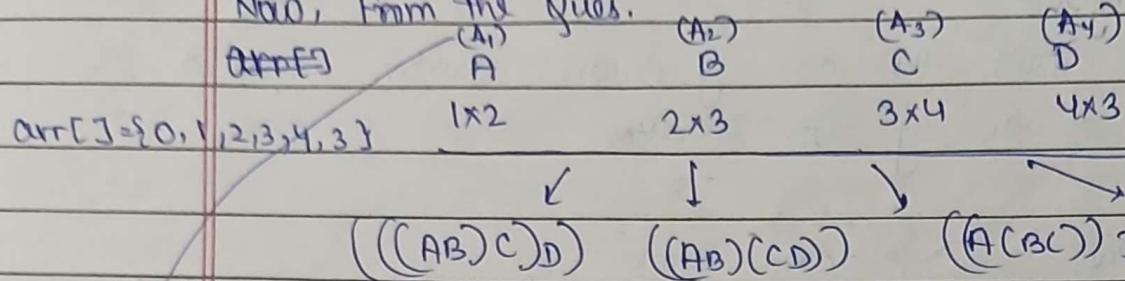
A1.A1: result = 1x6, cost = 1x3x6 = 18 : A2

Total cost = 90 + 18 = 108

min cost  $\Rightarrow$  Ans.

Teacher's Signature.....

Now, Form the Ques.



Therefore:-

- ①  $i$  = starting point
- ②  $j$  = ending point
- ③  $k$  = start  $\rightarrow$  end (traverse) from  $i$  to  $j$

$$A_1 = \text{arr}[0] \times \text{arr}[1]$$

$$A_2 = \text{arr}[1] \times \text{arr}[2]$$

$$A_i = \text{arr}[i-1] \times \text{arr}[i]$$

$\checkmark$  ①  $A_i = \text{arr}[i-1] \times \text{arr}[i]$  { $i \rightarrow$  starting from 1} (obvious reasons)

Now dividing into subproblems for help with recursion

$\therefore A_1, A_2, A_3, A_4, \dots, A_{n-1}$  mtlb chote problems

$\underbrace{A_1, \dots, A_k}_X \quad \underbrace{A_{k+1}, A_{n-1}}_Y$  ki modad et bade problems nikalo.

also for  $(AB)(CD)$   $\underbrace{AB}_{\text{set 1}} \quad \underbrace{CD}_{\text{set 2}}$   $\underbrace{A(B(CD))}_{\text{set 1 set 2}} \dots \rightarrow (X \cdot Y)$

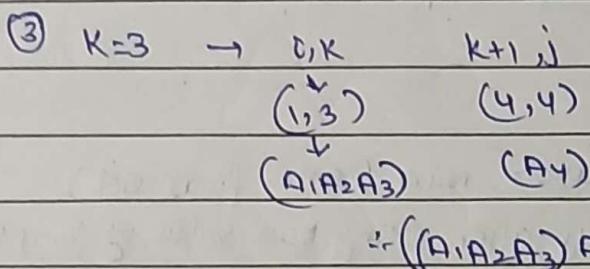
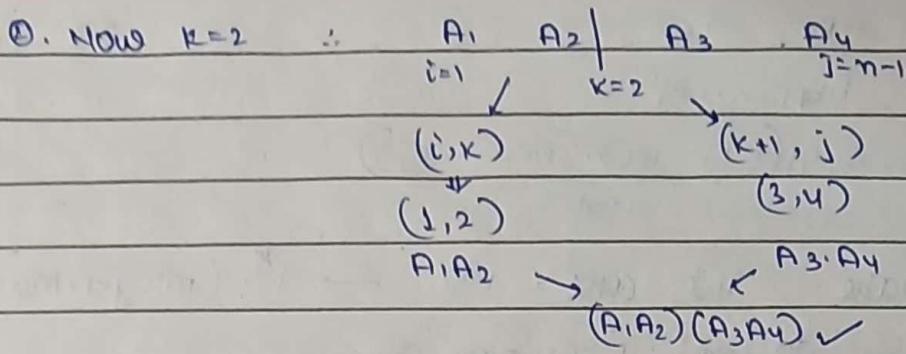
Approach  $\Rightarrow$  ①  $A_1 \mid A_2 \quad A_3 \quad A_4$

$\downarrow \quad \downarrow$   
 $(i, k)$   $(k+1, j)$   
 set set

$A_1 \quad \underbrace{A_2 A_3 A_4}_Y$

$\therefore A_1 \cdot Y$

Teacher's Signature.....



Hence, 'K' will traverse from  $i$  to  $j-1$ .

$\Rightarrow$  int mcm(arr[], int<sup>(i)</sup> i, int<sup>(j-1)</sup> j) {

for (int k=i to j-1)

cost1 = mcm(arr, i, k) \*

cost2 = mcm(arr, k+1, j) }

$\checkmark A_i = arr[i-1] \times arr[i]$

$A_k = arr[k-1] \times arr[k]$

$A_{k+1} = arr[k] \times arr[k+1]$

$A_j = arr[j-1] \times arr[j]$

$\checkmark result = [ \overbrace{arr[i-1] \times arr[k]}^{\text{row of } A_i} \overbrace{arr[k] \times arr[j]}^{\text{col of } A_k} ]$

also from this we get to know  $b=c$

$\checkmark$  Now if we see clearly multiplication of cost1 + cost2 is also valid. ( $b=c$ )

Therefore, ques. क्या hi set karke diya hogा jisse cost1 + cost2 multiplication feasible हो.

$\checkmark$  Cost 3 =  $a \times b \times d = arr[i-1] \times arr[k] \times arr[j]$

$\checkmark$  final cost = cost1 + cost2 + cost3.

✓ cost 1:  $A_i \dots A_k$

✓ cost 2:  $A_{k+1} \dots A_j$

✓ cost 3: cost of  $(\underbrace{A_i \dots A_k}, \underbrace{A_{k+1} \dots A_j})$   
cost 3

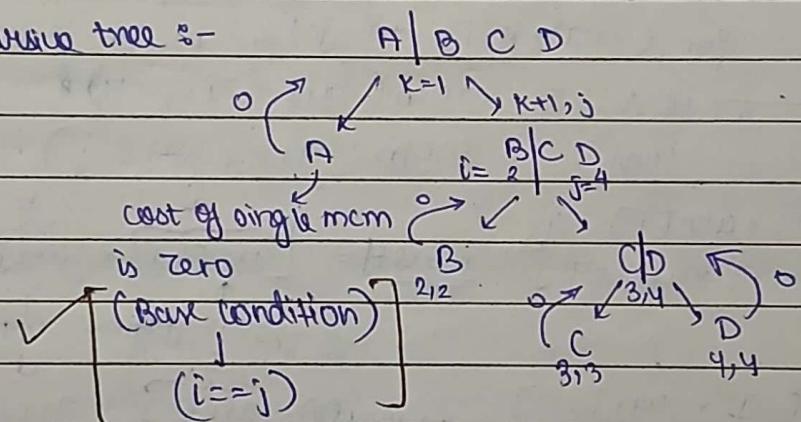
make int ans =  $+\infty \rightarrow$  i.e. (integer max value)  
to check & compare from  
final cost

initialise

We set answer(ans) ~~as~~ in loop  $\rightarrow$  ans = min (ans, final cost)  
to  $\infty$  so that to give ans as final cost only after getting least value of final cost.

(no need to check for condition (here), because it'll  
already be made out via Question).

recursion tree :-



Pseudo code  $\Rightarrow$  int mcm (arr, i, j) {

if ( $i == j$ ) {

return 0;

int ans =  $\infty$

for (int k = i to j-1) {

cost 1 = mcm (arr, i, k)

cost 2 = mcm (arr, k+1, j)

cost 3 = arr[i-1]  $\times$  arr[k]  $\times$  arr[j]

final cost = cost 1 + cost 2 + cost 3

ans = min (ans, final cost)

for first set as  $\infty$  bcoz ans. hamesha

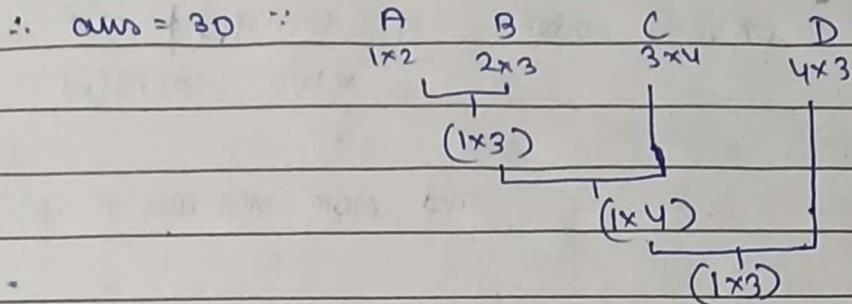
final cost walabi aaye

y return ans;

<sup>too</sup>, 108  $\times$  108 is stored in ans

Now, (108, 45)  $\rightarrow$  45 stored in ans

Teacher's Signature.....



$$1 \times 2 \times 3 = 6$$

$$1 \times 3 \times 4 = 12$$

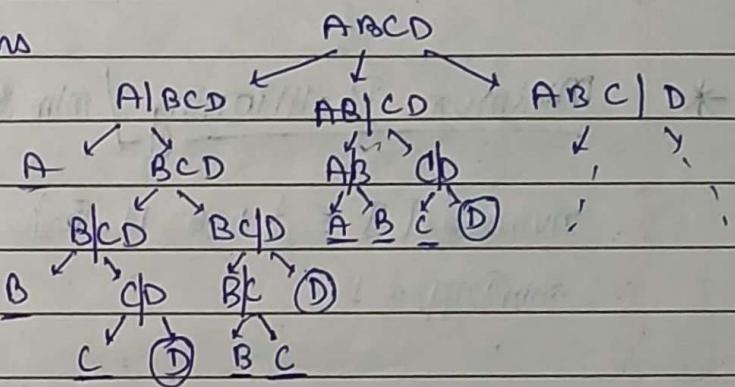
$$1 \times 4 \times 3 = 12$$

$$\underline{30 \text{ ans.}}$$

Final result (does not matter).

## Memoization approach

due to overlapping problems  
in recursion.



✓ Why DP?

→ ① choices

② overlapping substructure

✓ 2D Array since →  $i, j$   $\therefore \rightarrow A|BCD$  ( $i=1, j=\frac{n}{2}$ )  
 $(1 \text{ to } n)$   $\rightarrow (n-1, 1)$   $\therefore (i, k) \leftarrow A$   $\rightarrow (k+1, j)$   
 $1 \text{ to } (n-1)$ .  $(i=1, j=1)$   $(i=2, 4)$   $\therefore BCD$

✓  $\therefore dp[n][n]$  f fill with -1.  
 $i \downarrow$   $j \downarrow$   $i (1 \text{ to } n)$   $j (1 \text{ to } n)$

✓  $\text{arr}[ ] = \{ 1, 2, 3, 4, 3 \} \rightarrow n=5 \leftarrow 5 \times 5 \text{ matrix}$

✓ meaning of each block  $(i, j)$ . → e.g.  $(1, 2)$

$1 \times 2 \quad 2 \times 3 \quad \therefore 1 \times 2 \times 3 = 6$   
 $A_1 \quad A_2$   
 $(A) \quad (B)$

✓ Is bhi cell  $(i, j)$  mein hua, agar mai  $i \neq j$  tak matrix  $\downarrow$   
 multiply karu toh waha pe un matrix  $\downarrow$  multiplication  $\downarrow$  mtlb uska hai.

✓ changes from previous code :- if ( $dp[i][j] \neq -1$ )  
 return  $dp[i][j]$   
 }

and store min. ans in  $dp[i][j]$

✓ Time Complexity:  $O(n^2) \therefore O(n \cdot n)$ .

- Tabulation Approach  
 ↓

Intuitive

(∴ leave for now)

\* Minimum Partitioning / min. subset sum difference / Partitioning subsets

numbers[] = {1, 6, 11, 5}

minDiff = 1.

→ sum

⇒ aisa set choose karo jin do partition, aur difference min. aaye

$$\{11, 5\} \leftrightarrow \{1, 6\}$$

set1                  set2

$$16 \quad \quad \quad 7 \quad \quad \quad \therefore 16 - 7 = 9$$

$$\text{illy, } \{1, 11\} = 12 \quad \{6, 5\} = 11 \Rightarrow \textcircled{1} \rightarrow \text{min. diff.}$$

$$\{1, 5\} = 6 \quad \{6, 11\} = 17 \Rightarrow 11$$

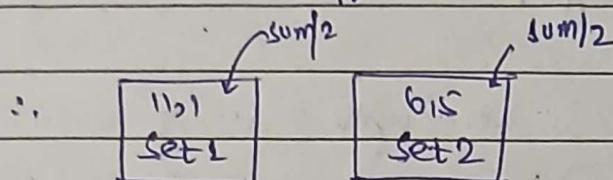
→ minimum difference will be when sum of set1 = sum of set2

$$| \text{sum2} - \text{sum1} | = \text{min.}$$

So in order to make this minimum we'll make sure to put  $\frac{\text{sum}}{2}$  elements <sup>or close to it</sup> in each set.

Sum  $\Rightarrow$  sum of all nos. i.e.  $1+6+11+15 = 33$

$$\therefore \text{sum}/2 \approx 16.5$$



also if we select elements for set 1  $\therefore$  remaining will automatically goes to set 2.

$\therefore$  fill set 1 only & put rest in 2.

$$\therefore \text{capacity}(w) = \frac{\text{sum}}{2} \quad \xrightarrow{\text{max. weight}}$$

and select/ignore element in such a way to get min. Difference

Variation of

O/I knapsack.

Also, when wt. & val are not given separately  $\therefore$  take  $\text{val} \Rightarrow \text{wt.}$

Valued.      Invalid.  
 take it      e.g.  $w=7$   
 ignore it      and nos. = {1, 12, 3}  
 $\therefore$  (min-Diff.)

Pseudocode  $\Rightarrow$  1. find sum of all nos.

(steps) 2.  $\text{dp}[n+1][w+1]$

$\downarrow$   
nums.length  $\rightarrow$  sum/2

here  $\text{dp}[5][12]$

$\downarrow$   
1

3. O/I knapsack

4.  $\text{dp}[n][w] = \text{sum } 2$

$\downarrow$   
 $23b = 11$

$$\text{sum } 2 = \text{sum} - \text{sum } 1$$

$$\text{min. Diff.} = \left| \text{sum } 1 - \text{sum } 2 \right|$$

Math.abs (sum1 - sum2).

Code → psvm {

int[] nums = {1, 6, 11, 5};

static int minPartition(int[] arr) {

int n = arr.length;

int sum = 0;

getsum {  
for (int i = 0; i < arr.length; i++) {  
sum += arr[i];  
}

int W = sum / 2;

int dp = new int[n + 1][W + 1];

for (int i = 1; i < n + 1; i++) {

for (int j = 1; j < W + 1; j++) {

if (arr[i - 1] <= j) { // valid

OR knapsack code ←       $dp[i][j] = \max(\text{arr}[i-1] + dp[i-1][j - \text{arr}[i-1]],$

$dp[i-1][j]);$

}  
else {

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

}  
}

↓  
ignore

int sum1 = dp[n][W];

int sum2 = sum - sum1;

return Math.abs(sum1 - sum2);

}

Teacher's Signature.....

- Minimum Array Jumps

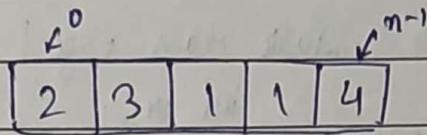
↓

To find min. jump to go from Start index ( $0^{th}$ ) to End- $n^{th}$  index ( $n-1$ ).

∴  $dp[i]$ :  $i^{th}$  index  $\rightarrow n-1$  take min. jump.

int arr = {2, 3, 1, 1, 4};

min Jumps = 2.



1 → 2 (Step=1) jumps =  $dp[2]+1$       choti problem (already calculated)

1 → 3 (" " = 2) " "       $dp[3]+1$

1 → 4 (" " = 3) " "       $dp[4]+1$

" going from  $n-1 \rightarrow n-1$  } smallest problem

$3 \rightarrow n-1$

:

$0 \rightarrow n-1$  } longest problem

∴ using  $n-2$  to 0 in for loop

∴  $n-1 \rightarrow n-1 = 0$  steps (base) (already calculated)

ans = 700

Pseudocode ⇒ for (int i = n-2 to 0)

int steps = arr[i]; → To bhi us index ki value shift utne steps liye jaa sakte hain.

j → traversal of steps till steps mentioned

for (int j = i+1; j <= i+steps && j < n; j++) {  
if (i=1 ∴ j=2, 3, 4)      i=1+steps(3)      this condition also  
= 4      ↓      needs to be satisfied  
max. of j can go      ∵ array  $\rightarrow$  bahan Kaha jaoge.

i = Kaha se Jump leua  
j = Kaha tak Jump leua

→ if ( $dp[j] \neq -1$ ) {

Also store -1 in array if  
Jump is not feasible.

ans = min (ans,  $dp[j]+1$ )

$dp[i] = ans (\neq \infty)$

e.g. [1/1/1/1/0]      1/0/0  
↑  
jab tak calc.  
nhi hua tab  
take already -1 hui

will be -1 since  
2 & tabhi pahuch hi nhi jaute.

∴ If all indexes not feasibles  
→ ans = -1.

return  $dp[0]$

∴ O & n-1  $\rightarrow$  kitne min. steps.

## GRAPHS - Important codes and points practice

refer theory notes from copy (main).

Store a graph → 1. Adjacency list 2. Adj. Matrix 3. Edge List  
 Preferred → 4. 2D matrix.  
 :: optimized

\* Adjacent list → static class Edge { } // static → to call in psvm without making its object,  
 ↓  
 making array of ArrayList  
 int src;  
 " wt;  
 " dest; } declarations

public Edge (int src, int dest, int wt) {  
 this.src = src;  
 this.wt = wt;  
 this.dest = dest; } constructor  
 ↓ to call it  
 in a fixed manner.

psvm {

int V = 5; // Vertices defined must define size

ArrayList<Edge>[] graph = new ArrayList[V]; → Array of A.L  
 for (int i=0; i < graph.length; i++) { declaration

graph[i] = new ArrayList<>(); → A.L added to all  
 } src dest wt index.

graph[0].add(new Edge(0, 1, 5)); → Since we're getting  
 size() from an ArrayList  
 my for 5 vertices.

to get neighbours → for (int i=0; i < graph[2].length; i++)  
 (e.g. 2)  
 Edge e = graph[2].get(i); // getting from Edge  
 sout(e.dest + " "); one by one

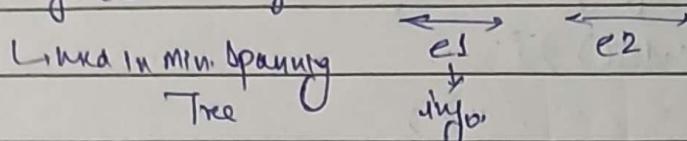
+ Do not forget to use util to get ArrayList.  
 ⇒ import java.util.\*;

Date		
Page No.		

- In Adj matrix we put same node on rows & col to check connections (1) for each. Since elements repeated for no we ∴ not optimised +  $O(V^2)$

Benefit → rather checking connections we can put weight there ↗

- Edge list → Edges = { {0,1,4}, {1,2,0}, {2,3,4}, ... }

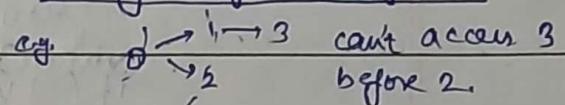


- Implicit → use rec. approach to fill matrix that contains info of graph.

### Graph Traversals

1. BFS → do horizontally level wise by checking neighbours first.

2. DFS (wait)



NOTE:- There's a theory of Wrapper class in ~~two~~ Graph traversals.

BFS → write boiler plate code for Graph (adj-list → same for all)

↓  
rather doing all ~~initialization~~ in psvm do it in ~~args~~ <sup>another func</sup>  
~~creation initialization~~  
+ create in psvm.

```
public static void vis(ArrayList<Edge>[] graph){}
```

```
Queue<Integer> q = new LinkedList<>();
```

```
boolean[] vis = new boolean[graph.length];
```

```
q.add(0); // Starter
```

```
while(!q.isEmpty()) {
```

```
int curr = q.remove(); // FIFO
```

```
if (!vis[curr]) {
```

```
cout(curr + " ");
```

```
vis[curr] = true;
```

```
for (int i=0; i<graph[curr].size(); i++) {
```

```
Edge e = graph[curr].get(i);
```

```
q.add(e.dest);
```

Teacher's Signature.....

In psvm → declare graph → call created graph → call bfs.

- 2. DFS  $\rightarrow$  priority on the basis of neighbour not on level.

```

code → public static void adj( ArrayList<Edge>[] graph, boolean[] vis, int curr) {
    always fixed ← { sout(curr + " ");
    vis[curr] = true;
    for (int i = 0; i < graph[curr].size(); i++) {
        Edge e = graph[curr].get(i);
        if (!vis[e.dest]) {
            dfs(" ", " ", " ");
        }
    }
}

```

- ## • HasPath?

```

code → public static boolean hasPath(graph, src, dest, [vis]) {
    if (src == dest) {
        return true
    }
    vis[src] = true;
    if (!vis[dest]) {
        return hasPath(graph, e.dest, dest, [vis])
    }
    return true;
}
return false

```