Dynamic Programming

Checklist :-

1. Memoization & Tabulation.
2. 1D DP Problems.
3. 2D DP Grid Problems.
4. Knapsack & Unbounded Knapsack Problems.
5. DP on String.
6. MCM problems & DP on Trees.

What is Dynamic Programming ?

- Misleading name
- Advanced / Optimised Recursion.

PreRequisites :-

1. Recursion ⟶ Basic, Understanding, multiple Calls.
2. Arrays, 2D Arrays ⟶ Basics.
3. HashMaps & Trees ⟶ Basics.

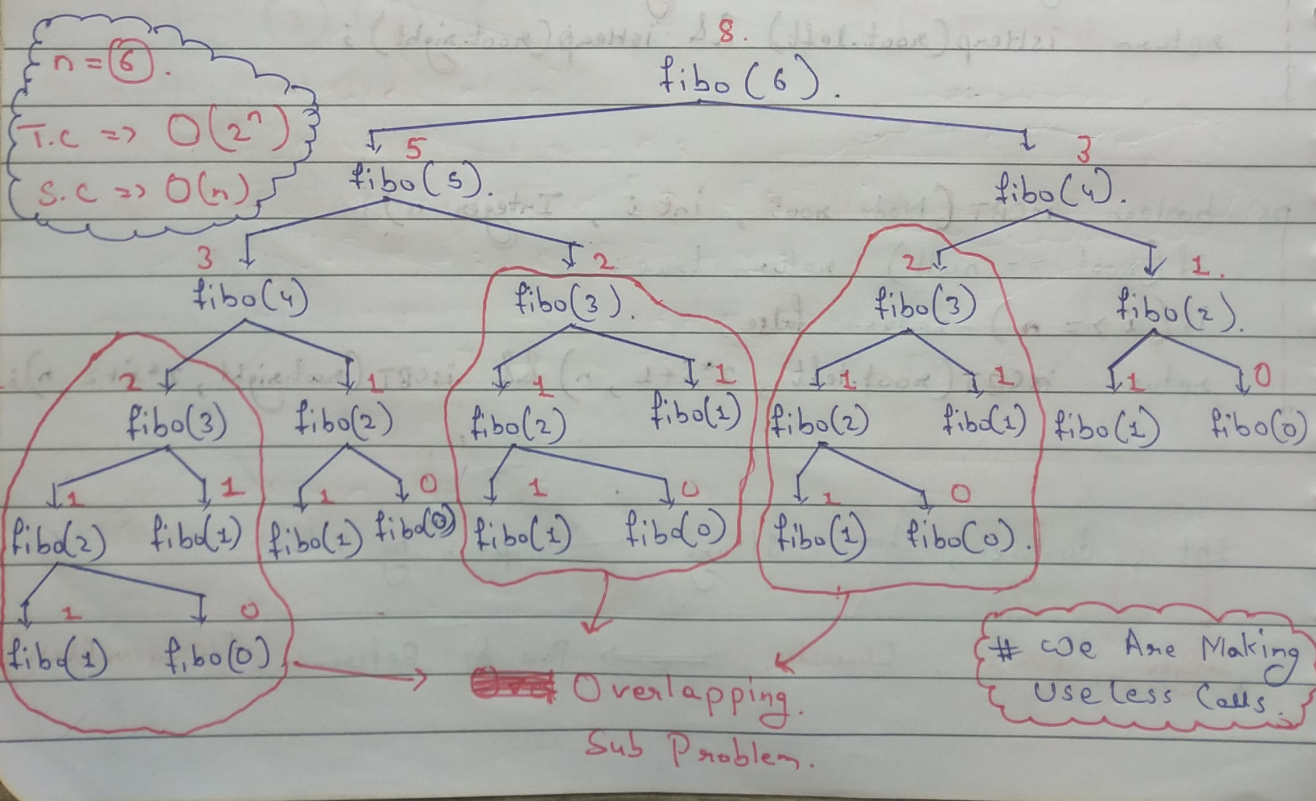
1. LeetCode Q. No. 509 { Fibonacci Number (Recursion) }

0 1 1 2 3 5 8 13 21 34 55 89 ...

n = 0 1 2 3 4 5 6 7 8 9 10 11

fibo(n) = fibo(n-1) + fibo(n-2).

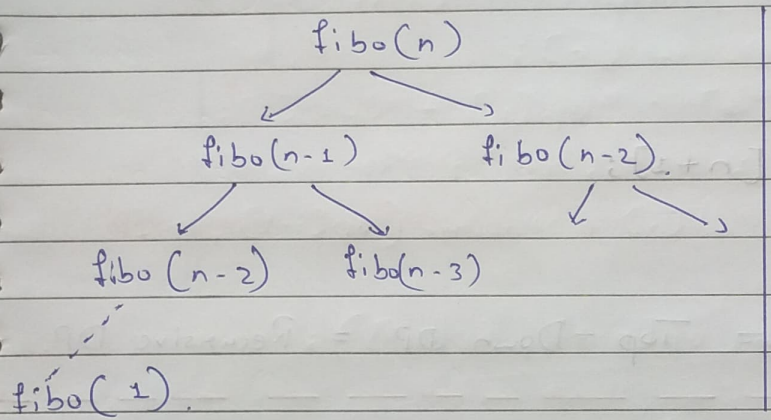
n = 6.
T.C => $O(2^n)$
S.C => O(n)



=> Code:-

```
public int fib(int n) {
    if (n <= 1) return n;
    return fibo(n-1) + fib(n-2);
}
```

=> Time Complexity Discussion.



For Calculating T.C. Count the total No. of Calls.

$1 + 2 + 4 + 8 \ldots 2^{n-1}$.

$\simeq (2^n)$ //

→ It is a balanced binary tree with 'n' levels.

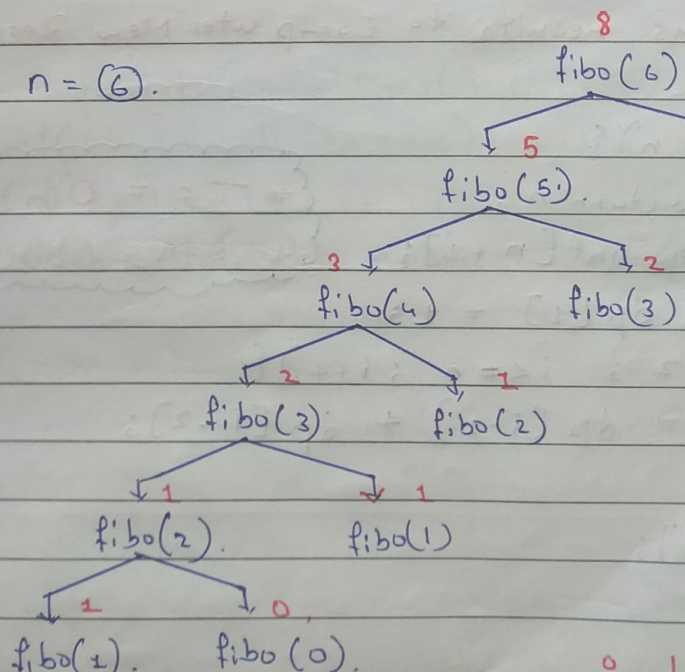
→ So, No. of Nodes / Calls Are Approximately (2^n) //

$O(\log n) > O(n) > O(n \log n) > O(n^2) > O(n^2 \log n) > O(n^3) >> O(2^n)$

Very Slow

Fibonacci Number (Recursion + Memoization):-

n = (6).



Time Complexity:-

Total Calls → $2n-1$ //

T.C = O(n).

S.C = O(n).

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

=> Code :-

```
public int fibo(int n, int[] dp){
    if(n <= 1) return n;
    if(dp[n] != 0) return dp[n]; // New.
    int ans = fibo(n-1, dp) + fibo(n-2, dp);
    dp[n] = ans; // New
    return ans;
}

public int fib(int n){
    int[] dp = new int[n+1];
    return fibo(n, dp);
}
```

=> Recursion + Memoization = Top-Down DP = Recursive DP.

Fibonacci Number (Tabulation).

↳ Bottom-Up DP = Iterative DP.

n = 8.

	0	1	2	3	4	5	6	7	8
dp =	0	1	1	2	3	5	8	13	21

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

Dp → Using previous results to Compute New Result.

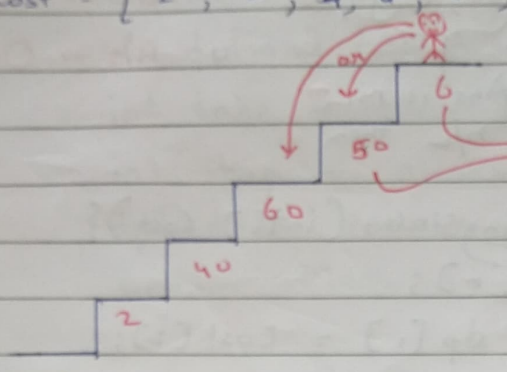
```
public int fib(int n){
    if(n <= 1) return n;
    int[] dp = new int[n+1];
    dp[0] = 0; dp[1] = 1;
    for(int i = 2; i <= n; i++){
        dp[i] = dp[i-1] + dp[i-2];
    }
    return dp[n];
}
```

T.C = O(n)
S.C = O(n)

② LeetCode Q.No. (746) { Min Cost Climbing Stairs. }

Page No.: YOUVA
Date:

Cost = [1 , 2 , 40 , 60 , 50 , 6]
(0) (1) (2) (3) (4) (5)



Can only take 1 or zero step.

we want Min^m Cost.

It can start from one of these only.

greedy (cost) = 6 + 50 + 40 + 1.
= 97 ✗

minCost = ~~66~~ 6 + 60 + 2
=> 68 ✓

=> Recursion [TLE]

=> Code:-

```
p int minCost (int[] cost, int idx){
    if (idx == 0 || idx == 1) return cost[idx];
    return cost[idx] + Math.min (minCost (cost, idx-1), minCost
                                 (cost, idx-2));
}

p int minCostClimbing Stairs (int[] cost){
    int n = cost.length;
    return min (minCost (cost, n-1), minCost (cost, n-2));
}
```

T.C = O(2^n). # A.S = O(n) → Recursive Stack Space

=> # Memoization:-

=> Code:-

```
pe int minCost (int[] cost, int idx, int[] dp){
    if(idx == 0 || idx == 1) return cost[idx];
    if(dp[idx] != -1) return dp[idx];
    return dp[idx] = cost[idx] + min (minCost (cost, idx-1, dp), minCost
                                      (cost, idx-2, dp));
}

p int minCostClimbingStairs (int[] cost){
    int[] dp = new int[n];
    Arrays.fill (dp, -1);
    return Math.min (minCost (cost, n-1, dp), minCost (cost, n-2, dp));
}
```

T.C => O(n). # A.S = O(n)

M T W T F S S

Page No.: YOUVA

Date:

⇒ Tabulation :-

Cost = { 1, 4, 100, 3, 3, 50, 6 }.

dp = { 1, 4, 101, 7, ~~8~~ 10, 57, (56) } → Ans.

T.C ⇒ O(n)

A.S ⇒ O(n).

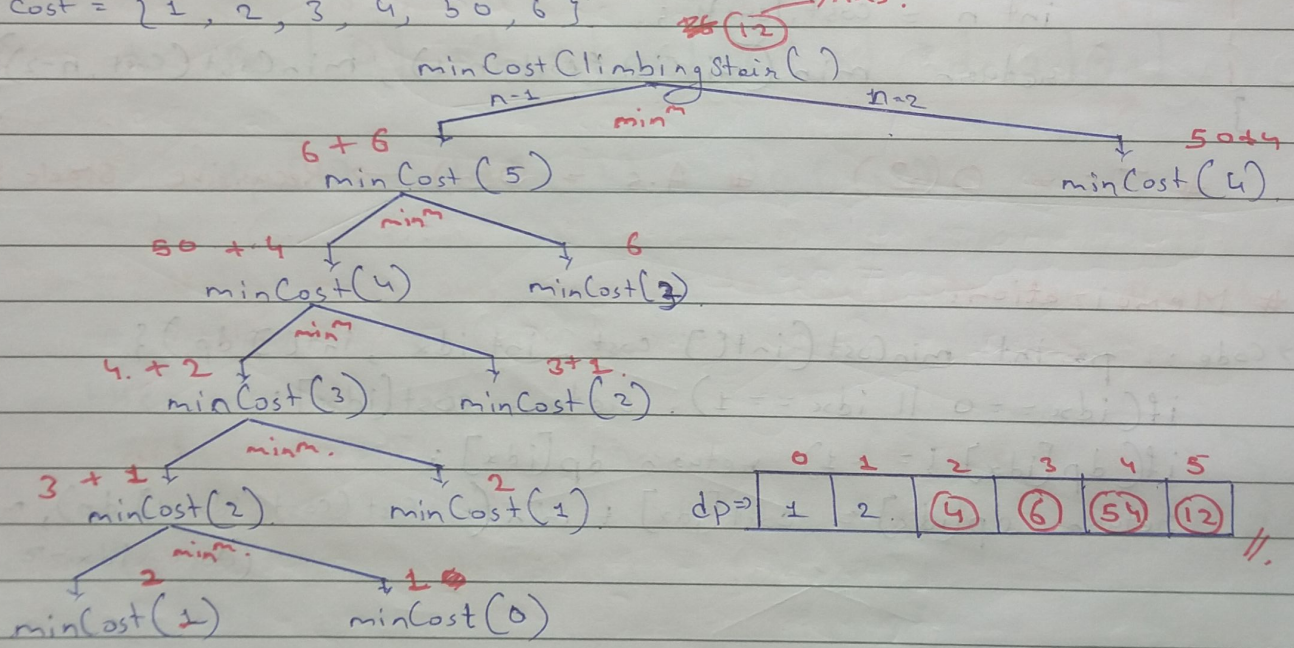
$dp[i] = cost[i] + \min(dp[i-1], dp[i-2])$ Ans is min^m b/w them

⇒ Code :-

```
int minCostClimbingStairs(int[] cost){
    int[] dp = new int[n];
    dp[0] = cost[0]; dp[1] = cost[1];
    for(int i = 2; i<n; i++){
        dp[i] = cost[i] + Math.min(dp[i-1], dp[i-2]);
    }
    return Math.min(dp[n-1], dp[n-2]);
}
```

Dry Run → Memoization

Cost = { 1, 2, 3, 4, 50, 6 }
(0 1 2 3 4 5)



dp ⇒

0	1	2	3	4	5
1	2	(4)	(6)	(54)	(12)

//.

③ Leet Code Q. No. (198) { House Robber } → Max^m Amount that can be ~~stolen~~ stolen.

arr = { 5, 15, 6, 20, 22, 1 }.

Greedy Approach ⇒ 5 + 6 + 22 = 33 ✗

15 + 20 + 1 = 36

Robber cannot ~~rob~~ Adjacent houses.

Max Amount ⇒ 15 + 22 ⇒ (37) Ans. //.

Memoization :- → Take & skip Algo.

M T W T F S S
Page No.:
Date:
YOUVA

⇒ Code :-
```
public int amount(int[] nums, int idx, int[] dp){
    if(idx >= nums.length) return 0;
    if(dp[idx] != -1) return dp[idx];
    int take = nums[idx] + amount(nums, idx+2, dp);
    int skip = amount(nums, idx+1, dp);
    dp[idx] = Math.max(take, skip);
    return Math.max(take, skip);
}

public int rob(int[] nums){
    int n = nums.length;
    int[] dp = new int[n];
    Arrays.fill(dp, -1);
    return amount(nums, 0, dp);
}
```

Tabulation :-

dp[0] = arr[0];
dp[1] = max(arr[0], arr[1]);

arr =	5	15	6	20	22	1

dp =	5	15	15	35	37	37

dp[i] = max(arr[i] + dp[i-2], dp[i-1]);
(arr[i] + dp[i-2] : take ; dp[i-1] : skip.)

dp[i] stores the max^m money that robber can rob from 0^th to i^th house.

⇒ Code :-
```
public int rob(int[] arr){
    int n = arr.length;
    int[] dp = new int[n];
    dp[0] = arr[0];
    if(n > 1) dp[1] = Math.max(arr[0], arr[1]);
    for(int i = 2; i<n; i++){
        dp[i] = Math.max(arr[i] + dp[i-2], dp[i-1]);
    }
    return dp[n-1];
}
```

(4) Friends Pairing Problem :-

M T W T F S S | Page No.: | Date: | YOUVA

You are given a number 'n'. Denoting that there are 'n' friends.

→ Find out the No. of ways in which a person can either pair up or stay single.

Ex :- a, b, c, d.

ways :-

- ⇒ ab, cd, ⇒ ac, bd ⇒ ad, bc
- ⇒ ab, c, d. ⇒ ac, b, d. ⇒ ad, b, c.
- ⇒ a, b, c, d ⇒ cd, a, b. ⇒ bd, a, c
- ⇒ bc, a, d.

Recursive Formula.

n = 3.

a, b, c

↓

- ⇒ a, b, c
- ⇒ ab, c
- ⇒ ac, b
- ⇒ bc, a

n = 4.

→ a, b, c, d.

#Single

a (b, c, d)

- a, b, c, d
- a, bc, d
- a, bd, c
- a, cd, b

#Pair

ab (cd)

- ab, cd
- ab, c, d

ac (db)

- ac, bd
- ac, b, d

ad (bc)

- ad, bc
- ad, b, c

⇒ Formula.

Generates all permutations. ab ≠ ba.

$$\#\,pairing(n) = n * [\,pairing(n-1) + (n-1) * pairing(n-2)\,]$$

pairing(n-1) → Single; (n-1) * pairing(n-2) → pair.

n → Ye Nahi Hoga

$$pair(4) = 4 * [\,pair(3) + 3 * pair(2)\,]$$

$$= 4[4+6] = 4[10] \Rightarrow 40 \text{ wrong.}$$

→ this is Correct.

$$pair(n) = pair(n-1) + (n-1) * pair(n-2)$$

Memoization :-

T.C ⇒ O(n)

S.C ⇒ O(n)

Tabulation :-

n = 6 :-

	0	1	2	3	4	5	6
dp =		1	2	4	10	26	76

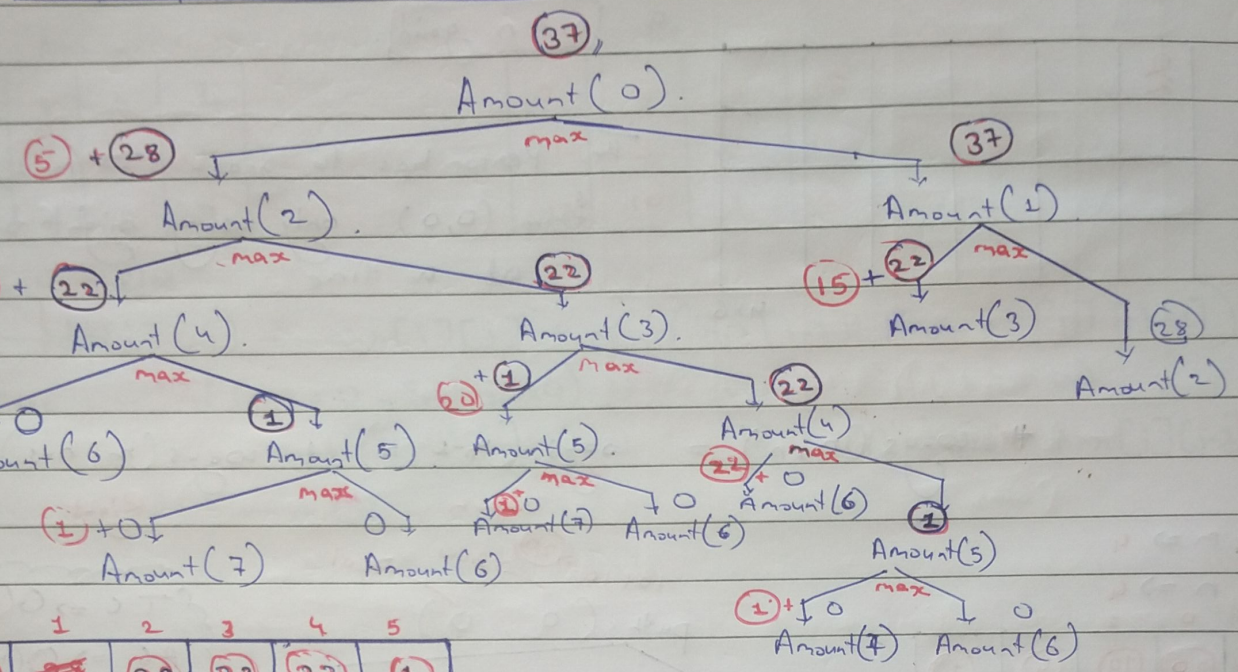
pair(n) = pair(n-1) + (n-1) * pair(n-2)

dp[i] = dp[i-1] + (i-1) * dp[i-2];

T.C ⇒ O(n) & A.S ⇒ O(n).

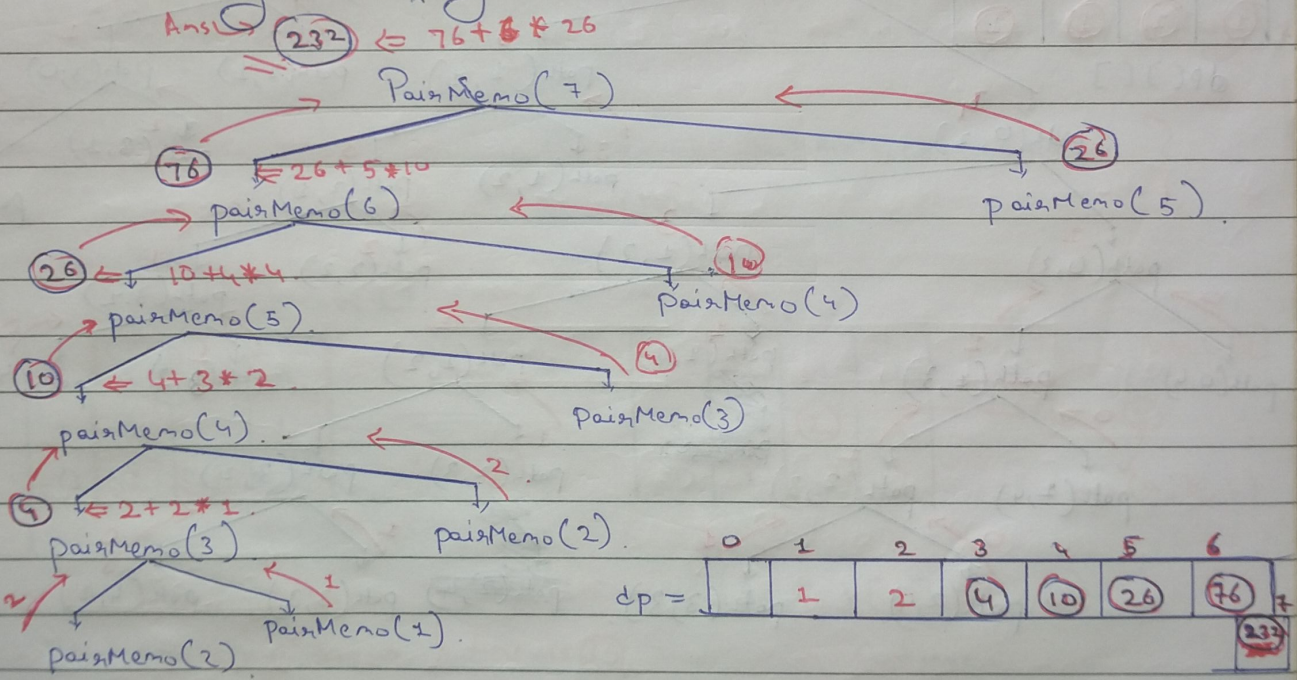
House Robber Tree Dry Run :-

0	1	2	3	4	5
5	15	6	20	22	1



	0	1	2	3	4	5
dP⇒	37	37	28	22	22	1

Friends Pairing Tree Dry Run :-

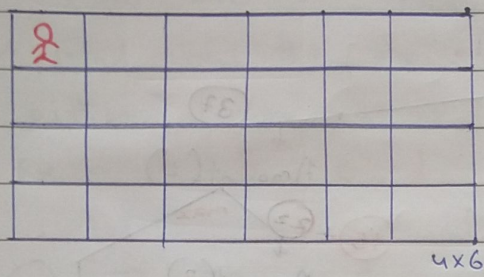


	0	1	2	3	4	5	6	7
dp =		1	2	4	10	26	76	232

(5) LeetCode Q. (62)

Unique Paths.

→ m & n where there is a m * n grid.



4x6.

The Person has to reach (m-1, n-1) from (0,0) only using right & down at a time.

ways(m, n) = ways(m-1, n) + ways(m, n-1);

down → Right

m ⇒ 4

n ⇒ 4

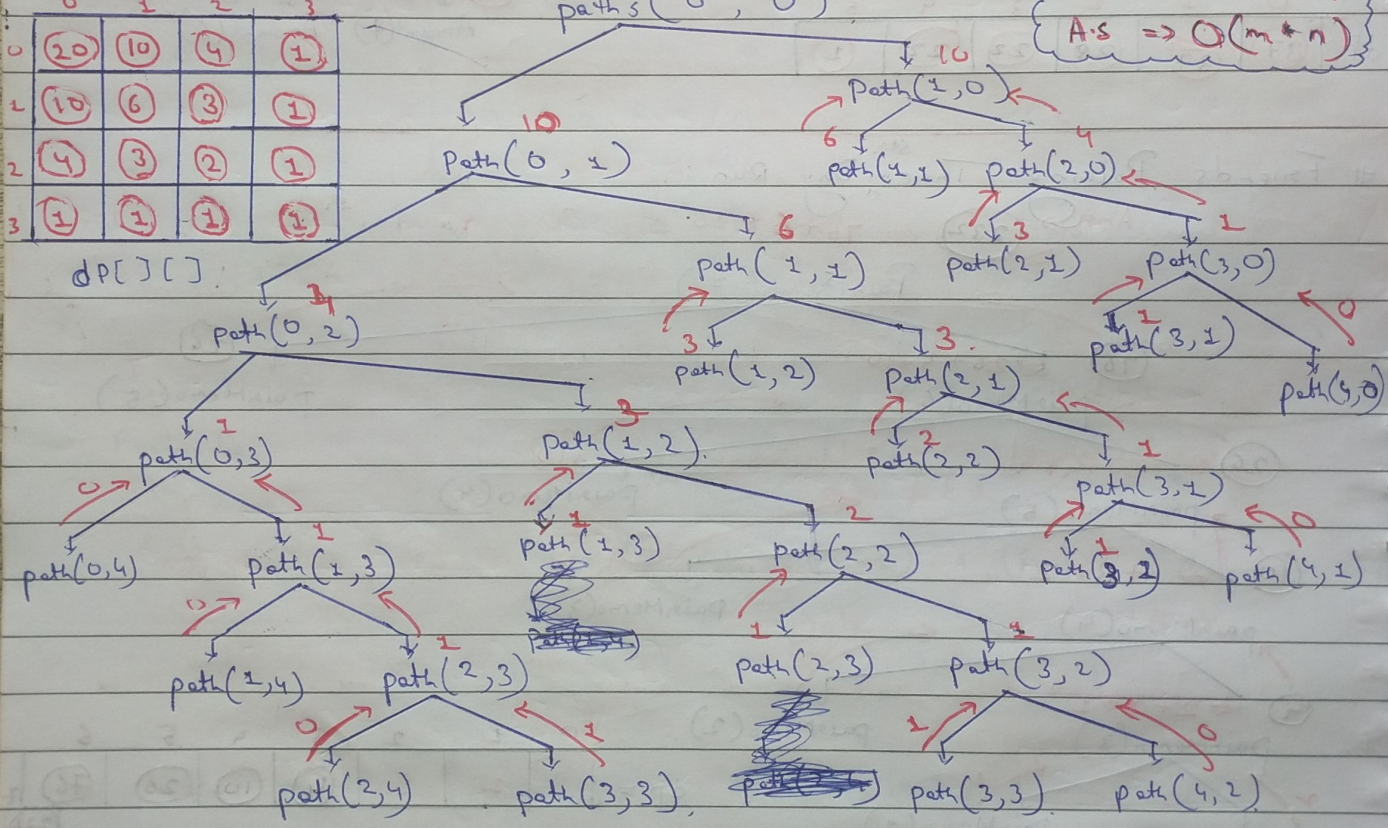
	0	1	2	3
0	20	10	4	1
1	10	6	3	1
2	4	3	2	1
3	1	1	1	1

dp[][]

10+10 ⇒ 20 → row, col.

T.C ⇒ O(m*n)

A.S ⇒ O(m*n)



Tabulation :-

	1	1	1	1	1
1	2	3	4	5	6
1	3	6	10	15	21
1	4	10	20	35	56

dp

4x6 → Ans.

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

T.C = O(m*n)

S.C = O(m*n)

⑥ LeetCode Q.No. (64) { Minⁿ Path Sum. }

M T W T F S S | Page No.: | Date: | YOUVA

Tabulation.

Arr

2	3	4	1
8	4	1	2
2	1	3	3

→ Minⁿ path.

Prefix Sum.

Dp

2	5	14	18
10	9	10	12
12	10	13	(15)

→ Ans.

Case 4:- else

dp[i][j] = arr[i][j] + min(dp[i-1][j], dp[i][j-1])

Case 1:- if(i==0 && j==0)

Continue;

Case 2:- if(i==0)

Arr[i][j] += Arr[i][j-1];

Case 3:- if(j==0)

Arr[i][j] += Arr[i-1][j];

Recursion:-

rightways = paths(row, col+1);

leftways = path(row+1, col);

return dp[row][col] = Arr[row][col] + Math.min(rightway, leftway);

⑦ LeetCode Q.No. (1277) { Count Square Submatrices with All Ones. }

Tabulation.

Arr

0	1	1	1
1	1	1	1
0	1	1	1

Size (1) Size (3). Size (2).

Arr.

0	1	1	1
1	1	(2)	2
0	1	2	3

Arr[i][j] = (1) + (1)

min(a, b, c)

=> 10 Squares of Size 1

=> 4 Squares of Size 2

=> 1 Square of Size 3

} Ans is (15)

if (Arr[i][j] == 0)

Continue.

else if (i>0 && j>0) {

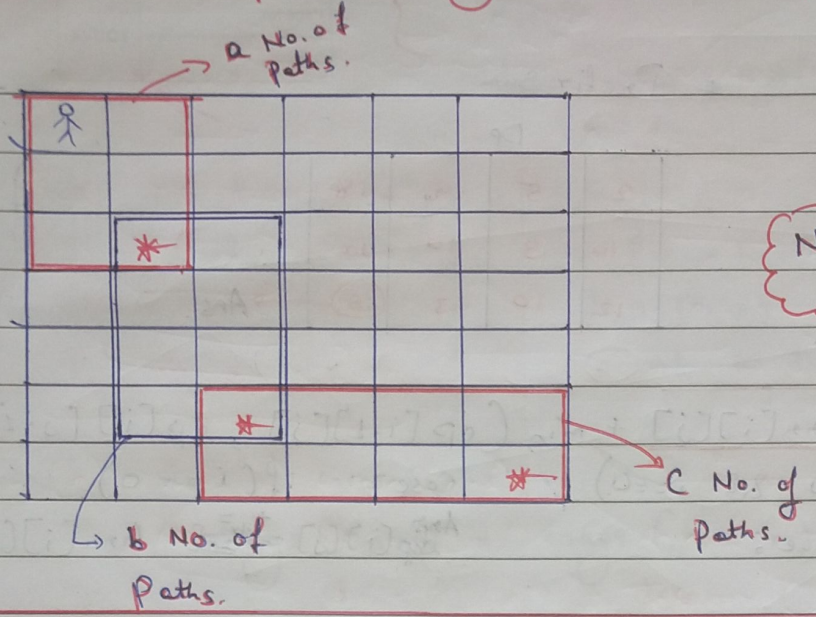
Arr[i][j] += min(a, min(b,c));

}

count += Arr[i][j];

{ a = Arr[i-1][j];
b = Arr[i][j-1];
c = Arr[i-1][j-1]; }

Paths which passes through Certain Checkpoints.



Right & Down.

No. of Paths => a * b * c.

Space Optimisation :- → [on Tabulation]

1D DP → 'n' size Array (dp) Space O(n) ⇒ O(1)

2D DP → 'm * n' size 2D Array (dp) Space O(m * n) ⇒ O(m) // O(n)

① Fibonacci Number (Tabulation) (Space Optimisation).

n = 8

n-1 Times loop

dp = | 0 | 1 | 1 | → | 1 | 1 | 2 | → | 1 | 2 | 3 |

| 2 | 3 | 5 |

| 3 | 5 | 8 |

| 5 | 8 | 13 |

| 8 | 13 | 21 |

Ans.

dp[2] = dp[0] + dp[1]

dp[0] = dp[1] // For Next

dp[1] = dp[2] // For Next

T.C = O(n)

S.C = O(1)

② Space Optimisation → Unique Paths. (Tabulation)

	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1	2	3	4	5	6
2	1	3	6	10	15	21
3	1	4	10	20	35	56
4	1	5	15	35	70	126

Optimisation

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

without Copy Pasting Also possible

	0	1	2	3	4	5
0	1	2	3	4	5	6
1	1	3	6	10	15	21

2 × n

	0	1	2	3	4	5
0	1	3	6	10	15	21
1	1	4	10	20	35	56

5 × 6. (m × n.)

Loop => (1 to m-1) {
  (1 to n-1) {
  }
}

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

	0	1	2	3	4	5
0	1	4	10	20	35	36
1	1	5	15	35	70	126

→ Ans.

⑧ LeetCode Q. No. 279 — Perfect Squares.

$n = a + b + c + \dots$

↓ ↓ ↓

Perfect Squares.

ans = min count of these

Ans

n = 3 → ③ = 1+1+3

4 → ① = 4

5 → ② → 1+4

6 → ③ → 1+1+4

7 → ④ → 1+1+1+4

8 → ② → 4+4

9 → ① → 9.

12 → ③ → 4+4+4

11 = 9+1+1

12 → 1 + (11) → 1+3 ⇒ 4

2 + 10

3 + 9

4 + 8

5 + 7

6 + 6.

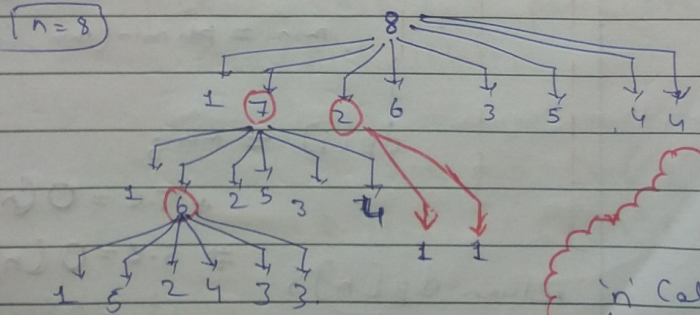
Recursion!-

numSquares(n) = numSquares(1) + numSquares(n-1)

numSq(2) + numSq(n-2)

T.C → <≈ $O(n^2)$

n = 8



NO. is Perfect square or Not

14 → √14 → 3

3*3 ⇒ 9

Not same

Not Perfect Square

Memoization

T.C ⇒ $O(n^2)$ → TLE

S.C ⇒ O(n)

'n' Calls.

but in every call → 'n/2' times for loop.

TLE

$i < n^2 \rightarrow i^2 < n$

Memoization

will be submitted.

Memoization with Optimization

M	T	W	T	F	S	S
Page No.:						YOUVA
Date:						

n = 12 ⟶ 1 + 11

⟶ 2 + 10

⟶ 3 + 9.

⟶ 4 + 8

⟶ 5 + 7

⟶ 6 + 6

$minS(n) \Rightarrow minS(i) + minS(n-i)$.

n = 12 ⟶ 1 + 11

⟶ $2^2 + (12 - 2^2)$ Or ⟹ (4) + 8

⟶ $3^2 + (12 - 3^2)$ Or ⟹ (9) + 3

we are making shure that ~~every~~ time we are sending At List on Prime No.

$minS(n) \Rightarrow minS(i^2) + minS(n - i^2)$.

T.C ⇒ $O(n\sqrt{n})$.

S.C ⇒ $O(n)$.

Tabulation :-

n = (7).

	0	1	2	3	4	5	6	7
dp ⇒		1	2	3	1	2	3	4 → Ans.

dp[4] → Prime No.

⇒ dp[2] ⇒ dp[1] + dp[1].

⇒ dp[3] ⟶ dp[1] + dp[2].

⇒ dp[6] ⟶ dp[1] + dp[5]

⟶ dp[4] + dp[2].

⇒ dp[5] ⟶ dp[1] + dp[4]

⟶ dp[4] + dp[1].

⇒ dp[7] ⟶ dp[1] + dp[6].

⟶ dp[4] + dp[3]

Code.

```
for(int j=1 to j<=i/2 ; j++){
    int count = dp[j] + dp[i-j];
    min = Math.min(count, min);
}
dp[i] = min;
```

T.C = $O(n^2)$

S.C = $O(n)$ It just work.

```
for( i=1 to i<=n ){
    if(isPerfect(i)) dp[i] = 1;
    else{
        int min = MAX_VALUE;
        for(int j=1 to j*j <= i){
            int count = dp[j*j] + dp
                [j*j-i];
            min = min(min, count);
        }
        dp[i] = min;
    }
}
return dp[n];
```

T.C = $O(n\sqrt{n})$.

S.C = $O(n)$.

It Obviously work.

0/1 Knapsack.

→ Bag

	Gold	IPhone	Table	Painting
price	5	3	7	16
weight	1	2	8	10.
Ratio ⇒	(5) max	(1.5)	0.8	1.6

(Capacity) C = (8).

Bag ⟶ Gold + iPhone
⟶ Profit = [8]

⇒ Objective ⟶ To Get Maxᵐ Amount (profit).

Capacity filled ⇒ 1+2 ⇒ (3)

Ring > painting
3 lakh, 10g — 10 lakh, 1kg.
More Valuable

Price / weight → Only in Fractional Knapsack.

→ This is Not Always Correct Ex.

⇒ Correct Logic :-

	Gold	IPhone	Table	Painting
price	5	3	9	16
weight	1	2	8	10.
Ratio ⇒	(5) max	1.5	1.1	1.6

C = 8.

Bag ⟶ Gold + IPhone
⟶ Profit = [8]
Capacity filed ⇒ (3)
#Wrong.

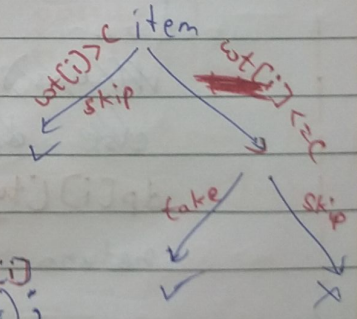
★ Ans :- [9]

Bag ⟶ Table
⟶ Profit ⇒ [9]
⟶ Capacity filed ⇒ (8)
#Correct.

[#Try Out All Possible Combinations]

⇒ Code :- # Recursion

```
ps int profit(int i, int[] wt, int[] val, int C){
    if(i == wt.length) return 0;
    int skip = profit(i+1, wt, val, C);
    if(wt[i] > C) return skip;
    int pick = val[i] + profit(i+1, wt, val, C-wt[i]);
    return max(pick, skip);
}
```



T.C ⇒ O(2ⁿ)
A.S ⇒ O(n*C)

Recursion + Memoization
* T.C ⇒ O(n*C)
* A.S ⇒ O(n*C)

(9) Subset Sum.

arr = { 0, 8, 5, 2, 4 } target = 9

Tell if there exists a subset of the array with sum == target.

Try Recursion & try All possible Subsets.
→ Pick or skip the Element.

Basic Recursion
TC = O(2^n);
A.S = O(n*t);

Recursion + Memoization
T.C = O(n*t);
A.S = O(n*t);

Recursion + Memoization:-

Code =>

```
p s boolean subset(int i, int[] arr, int target, int[][] dp){
    if(i == arr.length){
        if(target == 0) return true;
        else return false;
    }
    if(dp[i][target] != -1) return (dp[i][target] == 1);
    boolean ans = false;
    boolean skip = subset(i+1, arr, target, dp);
    if(target - arr[i] < 0) ans = skip; // Only Valid for +ve values.
    else {
        boolean pick = subset(i+1, arr, target-arr[i], dp);
        ans = pick || skip;
    }
    dp[i][target] = (ans) ? 1 : 0;
    return ans;
}
```

(10) LeetCode Q.No. (416) { Partition equal subset sum. }

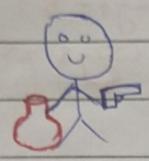
↳ Same Code like Previous One

Ex:-
nums = { 1, 5, 11, 5 }.

{ 1, 5, 5 }
{ 11 }
Sum is same
So true
All ele should be used.

Unbounded knapsack → Every item has Infinite Quantity

M T W T F S S | Page No.: | Date: | YOUVA



Ex ①

	Gold	IPhone	Table	Painting
Price	5	3	9	20.25
Weight	2	2	8	9
	2.5	1.5	1.1	2.25

C ⇒ 9.

maxProfit ⇒ 5 + 5 + 5 + 5 ⇒ 20 ✗
(2 2 2 2.)

→ 20.25 ✓
9.

Ex ②.

	Gold	IPhone	Table	Painting
Price	6	3	9	25
Weight	2	2	8	9.
	3	1.5	1.1	2.7

C = 9.

maxProfit ⇒ (25) ✓

0/1 knapsack	Unbounded
pick ⇒	i+1	i

(H) Leetcode (322) { Coin Change [Unbounded knapsack] } → Very Famous Question.

Ex ①:-

amount = (11). Coins = {1, 2, 5}.

(11) —5→ (6) —5→ (1) —1→ 0 // } → Greedy Method Not alway work.

Ans ⇒ 3 Coins

Ex ②.

[So. try All possibe Combination]

amount = (8) Coins = {1, 2, 4, 5}.

8 = 5 + 2 + 1 ⟶ 3 coins. ✗

8 = 4 + 4 ⟶ 2 Coins (Ans) ✓

(8) —5→ (3) —2→ (1) —1→ 0 // } → Wrong

knapsack General Code :-

```
if (amt - arr[i] < 0)   skip. → (i+1)
else {
    skip & pick
}.
```

'i' in Unbounded
'i+1' in 0/1

(12) LeetCode (494) { Target Sum }

arr = { 1, 1, 1, 1, 1 }

target = 3 → ways = 5.

t = 2 → ways = 0

+1 +1 +1 +1 -1 = 3.

+1 +1 +1 -1 +1 = 3.

+1 +1 -1 +1 +1 = 3.

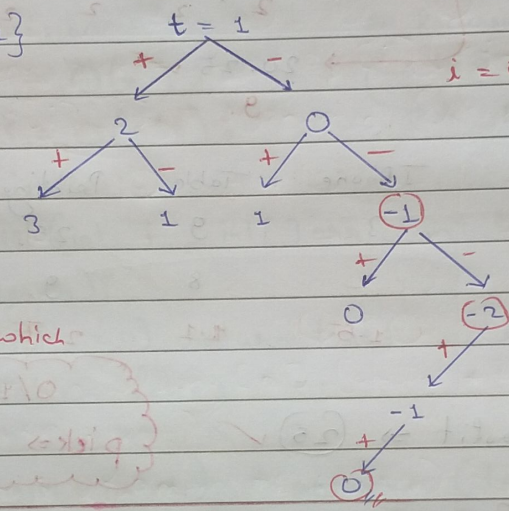
+1 -1 +1 +1 +1 = 3.

-1 +1 +1 +1 +1 = 3.

Knapsack → pick or skip strategy.

arr = { 1, 1, 1, 1, 1 }

target = 1.



dp[i][j] → stores the result of a call in which i & j

i = 0 to 4 target = 0 to t

dp[5][t+1];

target = -S to S

Transform 0 to 2*S

dp[n][2*S+1];

We can use hashmap for negative Indexing.

HashMap < Integer , HashMap < int , value > >

Integer → row number

int → -ve as well.

Tabulation :- 'in Knapsack Problem'

① Thinking of tabulation.

② Thinking of Recursion → Top-Down → Convert Bottom-up.

=> Steps :-

① idx → n-1 to 0 instead of 0 to n-1.

② Make formula.

③ Pls make sure of the base case

if (wt[i] > c) profit(i, c) = profit(i-1, c);

else profit(i, c) = max(val[i] + profit(i-1, c-wt[i]), profit(i-1, c))

⇒ dp[i][c] = max(val[i] + dp[i-1][c-wt[i]], dp[i-1][c]);

⇒ Code :- # Memoization

Top - Down.

```
if(i<0) return 0;
if(dp[i][c] != -1)
        return dp[i][c];
int skip = profit(i-1, c);
if(wt[i]>c)
        return dp[i][c] = skip;
else {
    int pick = val[i] + profit(i-1, c-wt[i]);
    return dp[i][c] = max(pick, skip);
}
```

Tabulation.

⇒ Code :-

Bottom - Up.

```
for (int i=0; i<n; i++){
    for (int j=0; j<c+1; j++){
        int skip = (i>0) ? dp[i-1][j] : 0;
        if (wt[i] > j)
            dp[i][j] = skip;
        else {
            int pick = val[i];
            pick += ((i>0) ? dp[i-1][j-wt[i]] : 0);
            dp[i][j] = Math.max(pick, skip);
        }
    }
}
```

Space Optimization In knapSack :-

⇒ Code :

```
long [][] dp = new long[n][amount+1];
for(int i=0; i<dp.length; i++){
    for(int j=0; j<dp[0].length; j++){
        long skip = (i>0) ? dp[i-1][j] : ((j==0) ? 0 :
                                          Integer.MAX_VALUE);
        if(j-coins[i]<0) dp[i][j] = skip;
        else{
            long pick = 1 + dp[i][j-coins[i]];
            dp[i][j] = Math.min(skip, pick);
        }
    }
}
```

⇒ Space Optimization :-

```
long [][] dp = new long[2][amount+1]; {i-1 : 0 | i : 1}
for(int i=0; i<n; i++){
    for(int j=0; j<amount+1; j++){
        long skip = (i>0) ? dp[0][j] : ((j==0) ? 0 :
                                        Integer.MAX_VALUE);
        if(j-coins[i]<0) dp[1][j] = skip;
        else{
            long pick = 1 + dp[1][j-coins[i]];
            dp[1][j] = Math.min(skip, pick);
        }
    }
    // Copy paste 1st row to 0th row.
    for(int j=0; j<amount+1; j++){
        dp[0][j] = dp[1][j];
    }
}
```

(13) LeetCode Q.No. (1143) { Longest Common Subsequence (Length) }.

M T W T F S S | Page No.: | Date: | YOUVA

Ex :-

s1 = bacdef (m)
s2 = aghcif. (n)

} ⟶ Ans => (3)
sacf

if(s1[m-1] == s2[n-1]) LCS = 1 + LCS (0 to m-2 of s1 & 0 to n-2 of s2)

s1 = bacdefx
s2 = aghcifb. } ⟶ {bacf ≠ acfb}.

ALGO :-

s1 = bacdef
s2 = aghcif

bacde + 1
aghci

max(bacde aghc , bacd aghci)

bacde
aghc

bacd e
aghc

bacde agh

bacd
agh

bacd
aghc

bac
agh

bac
aghc

ba
agh

ba + 1

Recursion + Memoization

T.C ⇒ O(m*n)
A.S ⇒ O(m*n)

Tabulation

⟶ We will understand tabulation in 2 ways :-

1. Just convert memoization to tabulation.

T.C ⇒ O(m*n)
A.S ⇒ O(m*n)

⟶ dp[m-1][n-1] ⇒ Stores LCS of substr(a, 0, m-1) & substr(b, 0, n-1).

⟶ dp[i][j] will store the LCS of substring of 'a' from 0 to i & substring of 'b' from 0 to j.

O(n) [Space Optimization]

* To Avoid the base case in conversition step, you can ~~read~~ make 'dp' larger ⟶ (m+1 * n+1).

(2) Thinking Via Tabulation :-

dp. (a) g h (c) i (f) → {we will only store No.}

		a	g	h	c	i	f
	b	0	0	0	0	0	0
(a)	"a" 1	"a" 1	"a" 1	"a" 1	"a" 1	"a" 1	
(c)	"a" 1	"a" 1	"a" 1	"ac" 2	"ac" 2	"ac" 2	
d	"a" 1	"a" 1	"a" 1	"ac" 2	"ac" 2	"ac" 2	
e	"a" 1	"a" 1	"a" 1	"ac" 2	"ac" 2	"ac" 2	
(f)	"a" 1	"a" 1	"a" 1	"ac" 2	"ac" 2	"acf" 3	

dp[i][j]

LCS { → s1 → 0 to i substr
      → s2 → 0 to j substr.

		0	1 (a)	2 (g)	3 (h)	4 (c)	5 (i)	6 (f)
	0	0	0	0	0	0	0	0
(b)	1	0	0 (max)	0	0	0	0	0
(a)	2	0	1 (0+1)	1	1	1	1	1
(c)	3	0	1	1	1	2 (1+1)	2	2
(d)	4	0	1	1	1	2	2	2
(e)	5	0	1	1	1	2 (max)	2	2
(f)	6	0	1	1	1	2	2	3 (2+1)

---

(14) Leetcode Q.No. (516) { Longest Palindromic Subsequence (length) }

Hint => (1) LCS
(2) Reverse of given string.

str = a b c d b f g a g
rev = g a g f b d c b a
→ LCS. Ans => (5)

LPS (str) => LCS (str , revStr)
n n

{ # T.C => O(n²)
A.S => O(n²) }

---

(15) Leetcode Q.No. (1312) { Minimum Insertion Step to make String Palindrome }

s => m b a d m → Ans (2)

(1) m b d a d b m → mbdadbm (2)
(2) m d b a b d m → mdbabdm

=> n => s.length() ;

Ans => n - lcs (s , revS) ; → for deletion also same code & Ans.

(H.W)

(16) Leetcode Q.No. (583) { Delete Operation for 2 String }

Page No.: / Date: YOUVA

Ex (1) :- S1 => "sea" S2 => "eat".

1st step delete "s" from S1.
2nd step delete "t" from S2. } → Now both the str's are exactly same. So. Ans => (2) steps.

Ex (2) :-

S1 => "leetcode", S2 => "etco"

Ans => (4) // [Need to delete "lede" from S1 //]

Ans => (m - lcs) + (n - lcs) → [lcs of s1 & s2 //]

→ Length of S1 // → Length of S2 //

---

(17) Leetcode Q.No. (72) { Edit Distance }

Ex (1) a = "movie"
b = "love"

Operations → only of 1st string.
- Delete
- Insert → 1st Don't think for it
- Replace

(1) Delete "i".
a => "move"
b => "love"

(2) Replace "m" with "l".
a = "love"
b = "love" → Now both are same //

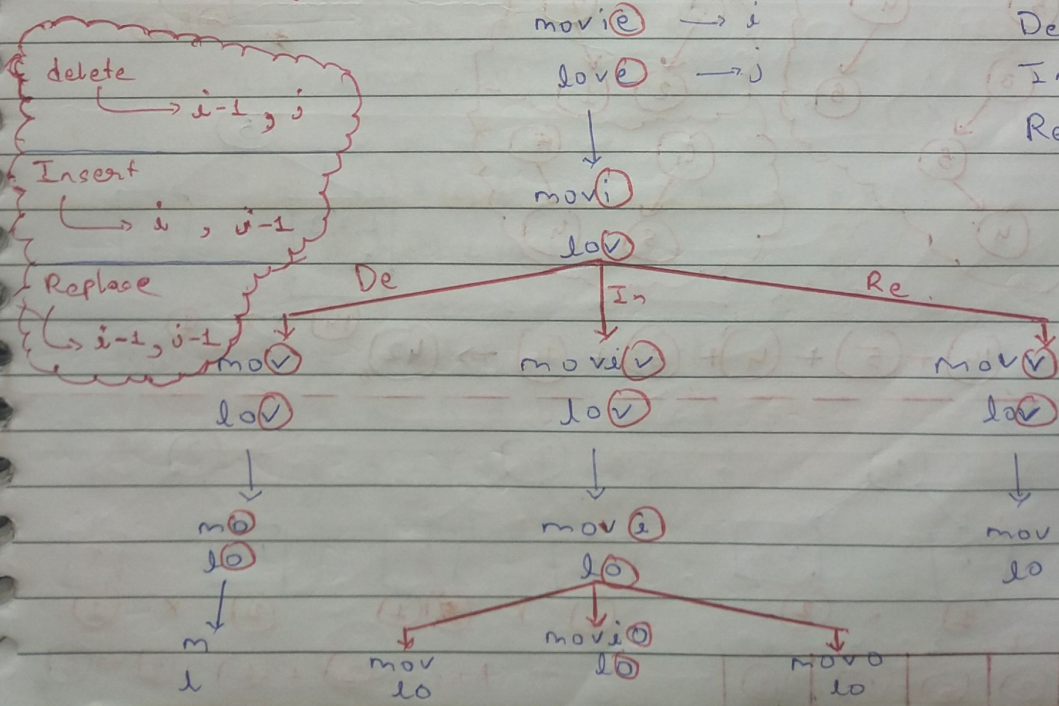
=> Ans => (2) //.

Recursion (Algo).

delete → i-1, i
Insert → i, i-1
Replace → i-1, i-1

movie → i
love → j

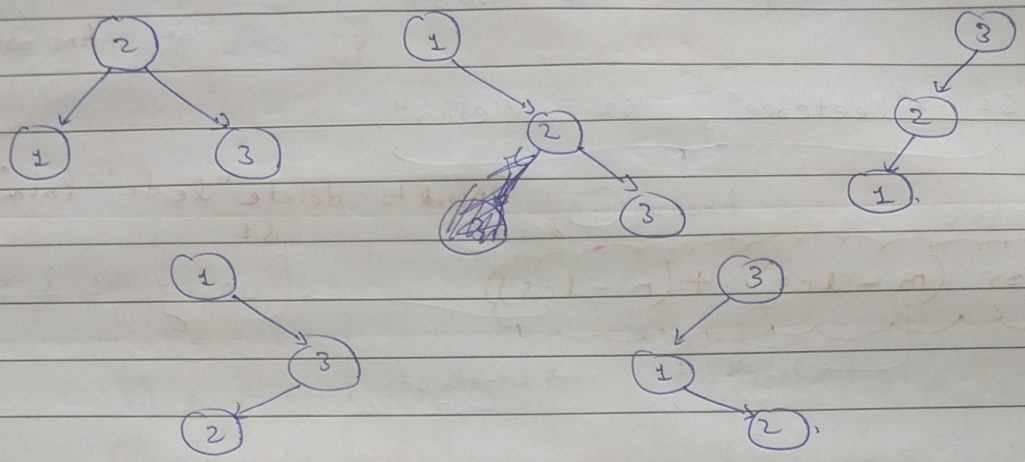
Delete
Insert
Replace



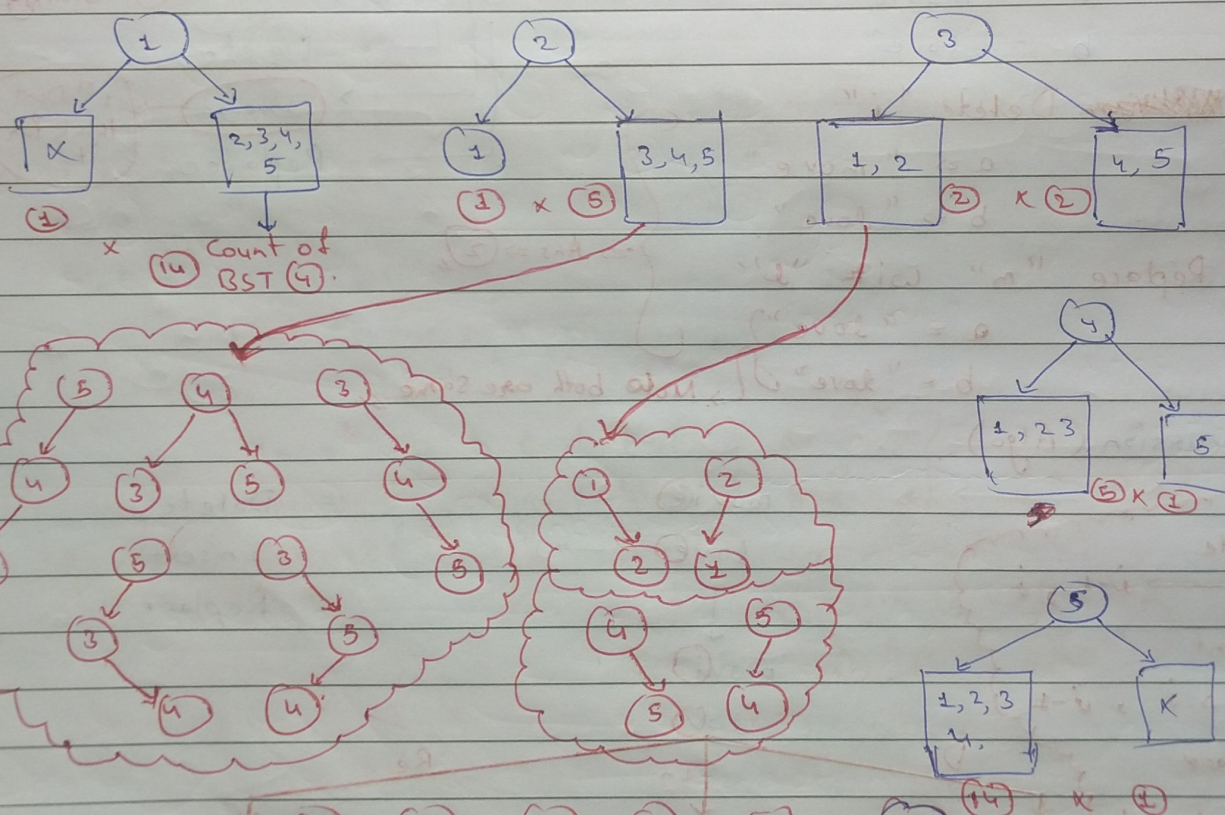
(13) LeetCode Q.No. (96) { Unique Binary Search Trees. }

M T W T F S S | Page No.: | Date: | YOUVA

n = 3 ⇒ Max^m Structures :-

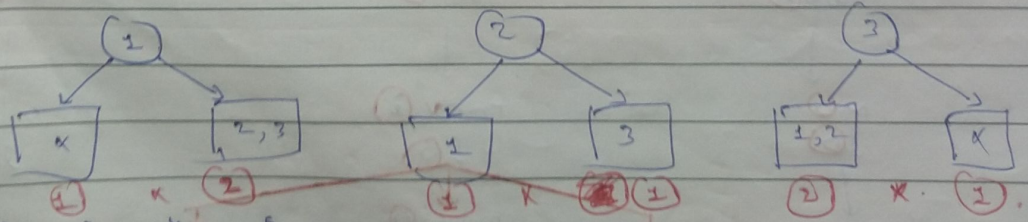


n = 5

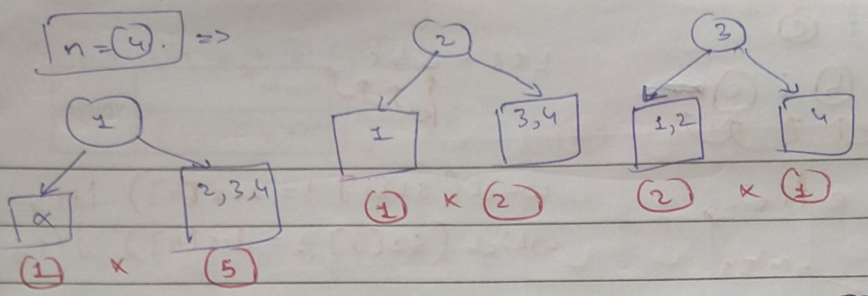


Ans ⇒ (14) + (5) + (4) + (5) + (14) ⇒ (42)

n = 3



Ans ⇒ (2) + (1) + (2) ⇒ (5)



Ans => 5 + 2 + 2 + 5 => 14

=> Code :-

```
pubic int numTrees(int n){
    if(n<=1) return 1;
    int[] dp = new int[n+1];
    dp[0] = 1 ; dp[1] = 1 ; dp[2] = 2;
    for(int i = 3 ; i < n+1 ; i++){
        for(int j = 1 ; j <= i ; j++){
            dp[i] += dp[j-1] * dp[i-1];
        }
    }
    return dp[n];
}
```

T.C => $O(n^2)$

A.S => $O(n)$

---

(19) LeetCode Q.No. (1092) { Shortest Common Supersequence } → string (length)

Ex :-

s1 = "abac" , s2 => "cab" → lcs => "ab"

scs => "cabac" ← Ans

scs => ~~"abacab"~~ Not shortest

1st → Find the length of Shortest Common Supersequence of s1 & s2

→ Think about lcs of s1, s2.

lcs + (s1 - lcs) + (s2 - lcs)

Ex :-

s1 = "a d b e c f , lcs = abc

s2 = g a h b i c

scs => g a d h b e i c f

g a h d b i e c f

g a d h b i e c f

g a h d b e i c f

Ex :- s1 = a d b e f c.

s2 = z g a h b i c

scs = z g a d h b e f i c.

lcs = a b c.

while (s1[i] != lcs[k]) i++;

while (s2[j] != lcs[k]) j++;

Ex :- s1 = a b a c → miss

s2 = c a b

lcs => a b

scs => c a b a c

Print Longest Common Subsequence

s1 => a b c d e → m

s2 => f a b g d k → n

lcs => a b d.

→ Need to print this

		0	1 (f)	2 (a)	3 (b)	4 (g)	5 (d)	6 (k)
0		0	0	0	0	0	0	0
1	a	0	0	1	1	1	1	1
2	b	0	0	1	2	2	2	2
3	c	0	0	1	2	2	2	2
4	d	0	0	1	2	2	3	3
5	e	0	0	1	2	2	3	3

Break ← 0

i => m, j = n.

(m+1)*(n+1)

```
if(a.(i-1) == b.(j-1)){
    ans.append(a.(i-1));
    i--; j--;
}
else{
    if(dp[i-1][j] >= dp[i][j-1])
        i--;
    else j--;
}
```

Ans => "d b a" → reverse → "a b d"

(20) LeetCode Q.No (647) { Palindromic Substrings. } (Tabulation)

s = gabcbab

→ g, a, b, c, b, a, b

→ bcb, bab,

→ abcba.

(i) Method-1 :- Brute Force → Generate All substring & check Each substring.

For Ex :-

s = abcba n = 5

→ a, ab, abc, abcb, abcba.

T.C => O(n^3)

1 + 2 + ... n + 1 + 2 ... n-1 + 1 + 2 ... n-2

$\sum_{k=1}^{n} k = \frac{k(k+1)}{2}$

1 + 2 + 3 ... k = $\sum_{k=1}^{} \frac{k(k+1)}{2}$ ≈ $\sum k^2 + k$

$\sum_{1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \Rightarrow n^3$

s = a b c b a
i ↑ j ↑

s[i] == s[j] && (i+1, j-1) substring is ~~palindrome~~

Page No.: YOUVA
Date:

Ex :-
s => g a b c b a b
0 1 2 3 4 5 6

	0	1	2	3	4	5	6
0	1	0	0	0	0	0	0
1	x	1	0	0	0	1	0
2	x	x	1	0	1	0	0
3	x	x	x	1	0	0	0
4	x	x	x	x	1	0	1
5	x	x	x	x	x	1	0
6	x	x	x	x	x	x	1

2-length Substring Palindrome kab hoti hai?
↓
Jab & Ch of 'i' & 'j' is same

Every cell stores true/1 if substring ~~from i to j~~ is palindrome

T.C => $O(n^2)$
A.S => $O(n^2)$

---

(21) {Gfg. Longest Common Substring (Tabulation)}

Ex :- s1 = abcdef
s2 = agcdhf

→ LCS => "acdf" → (4)
→ Longest Common Substring* = "cd" → (2)

Think Tabulation Method of longest Common Subsequence.

s1 = a b c d e f
s2 = b c a d e f

if(s1[i] == s2[j]) i, j → i-1, j-1

else i, j → max[(i-1, j), (i, j-1)]

a b c d e
b c a d e
↓1
a b c d
b c a d
↓1

Wrong for Substring
↓ Correct
0

a b c
b c a
a b c
b c
↓1

a b
b c a

S1 => a b * d e f

S2 => a b c d e f

		a	b	c	d	e	f
	0	0	0	0	0	0	0
a	0	0+1 → 1	0	0	0	0	0
b	0	0	1+1 → 2	0	0	0	0
d	0	0	0	0+1 → 1	0	0	0
e	0	0	0	0	1+1 → 2	0	0
f	0	0	0	0	0	2+1 → 3	0

→ Ans is Max^n.

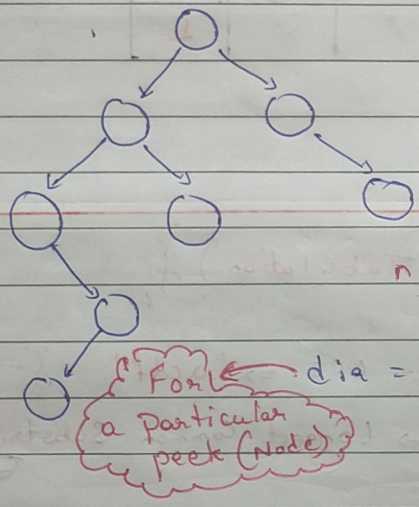
→ Ans => "def"

(m+1)(n+1)

---

(22) Leetcode (543) { Diameter of a Binary Tree } => Longest Path

Max^n = Diameter



path length = n - 1

↓

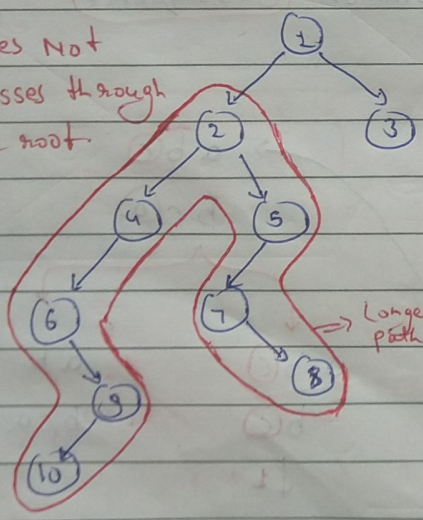
No. of Nodes in the path

n = 1 + levels(left) + levels(right)

-1

dia = levels(left) + levels(right).

For a particular peek (Node)



does Not passes through the root

Longest path

* Recursion

↳ T.C => O(n^2)

because for every Node we are calling levels → O(n)

=> Recursion + Memoization => Not Best

* We Can Memoise Using HashMap

map <TreeNode, Integer> dp;

node levels

↳ T.C => O(n).

↳ A.S => O(n).

Best Method → [Calculate ~~path~~ diameter ~~as~~ during the level code Once.]

↳ T.C => O(n)

↳ A.S => O(h)

↳ recursion stack.

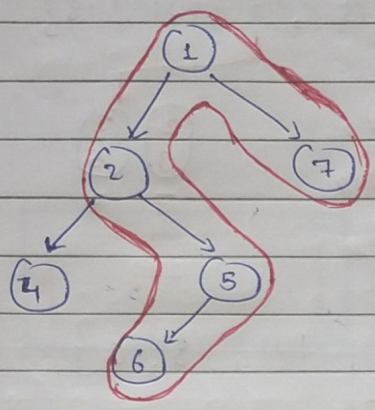
(23) LeetCode Q.No. (124)
Hard

Binary Tree
Maximum Path Sum.

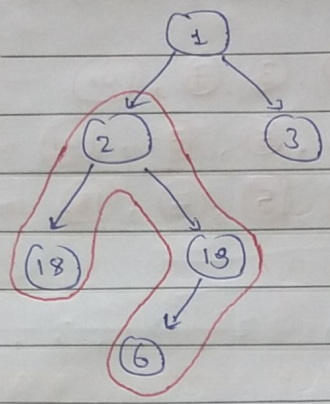
M T W T F S S

Page No.:	YOUVA
Date:	

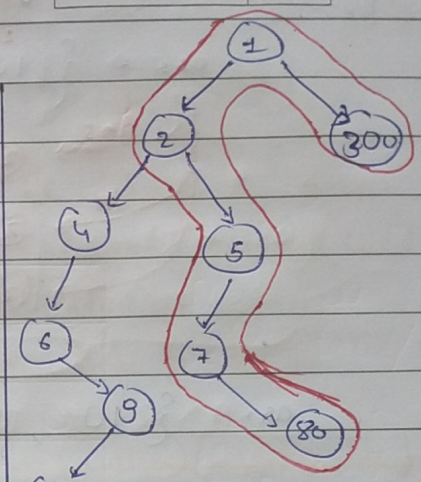
- Diameter of BT.



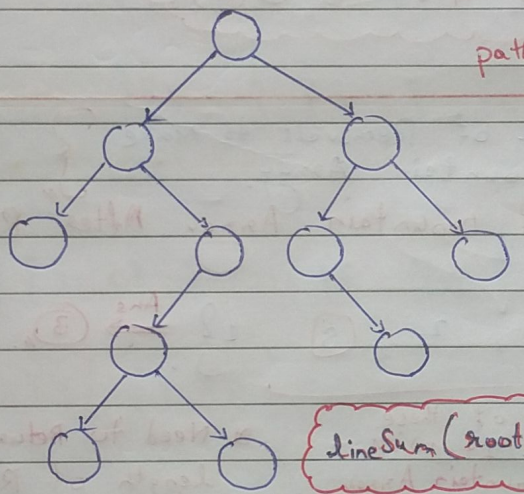
6 + 5 + 2 + 1 + 7 ⇒ (21)



18 + 2 + 19 + 6
⇒ (45)



⇒ As of Now we are Considering leaf to leaf.



root → leaf

pathSum = root.val + lineSum(left) + lineSum(right).

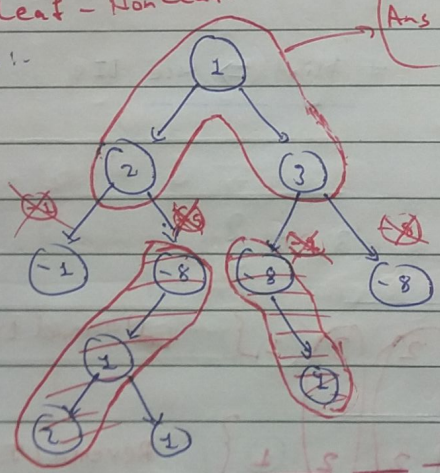
↳ lineSum ko 'maximise'.

maxSum = max(pathSum, maxSum);

lineSum(root) = root.val + max(lineSum(left), lineSum(right))

#Leaf - Non Leaf.

Ex:-

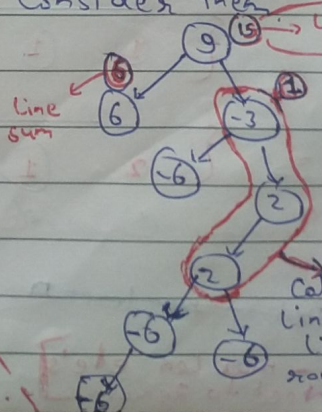


Ans ⇒ 1 + 2 + 3
⇒ (5)

Conside Node (2) & (3)

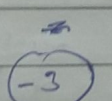
↳ if left lineSum or right lineSum is -ve then we will Not Consider them

Ex:-



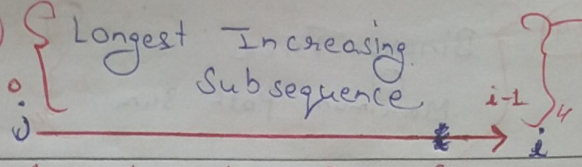
6+9
lineSum
Wrong
Out ⇒ 15
Exp ⇒ 16.
[6+9+1]
We need to calculate max lineSum Not lineSum from root to leaf

Ex :-



Ans ⇒ (-3)
Because we need to take Non-empty path means At least One needed.

(24) LeetCode Q.No. (300) { Longest Increasing Subsequence



Ex :-

arr = { 10 , 9 , 2 , 5 , 3 , 7 , 101 , 18 }

1 1 1 2 2

Ans → (2) , (3) , (7) , (101) ⇒ (4)

(2) , (5) , (7) , (101) ⇒ (4)

(2) , (5) , (7) , (18) ⇒ (4)

(2 3) (2 5)
7 7

arr = { 10 , 9 , 2 , 5 , 3 , 7 , 101 , 18 , 6 }

dp = { 1 , 1 , 1 , 2 , 2 , 3 , 4 , 4 , 3 } → Ans is: Max in this.

```
for(i=0 ⟶ n-1){
    for(j=0 ⟶ i-1){
    }
}
```

T.C ⇒ O(n²)
A.S ⇒ O(n)

---

(25) LeetCode Q.No. (1671) { Min^n No. of Removals to make Mountain Array. }

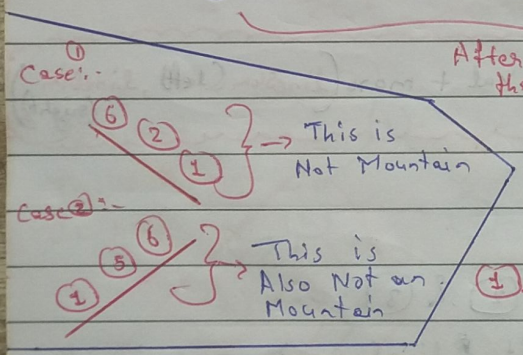
↳ Largest possible length of mountain Array After Removals.

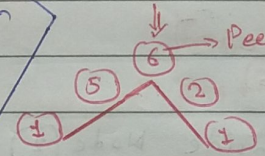
Ex :- { (2) , 1 , (1) , 5 , 6 , 2 , (3) , 1 } Ans ⇒ (3)

After Removal of these three Ele, It will be mountain Array.

Need to Return min^n length of Removal elements.

Case ① :-





Hint ⇒ use LIS

arr = { 2 1 1 5 6 2 3 1 }

LIS

dp1 = { 1 1 1 2 3 2 3 1 } → Normal LIS

LIS.

dp2 = { 2 1 1 3 3 2 2 1 } → Reverse LIS

we don't want # Case ① & ② like mountain so if any one of dp[i] has one don't Consider it.

2+3 → 5-1 → 4
2+3 → 6-1 → 5
2+2 → 4-1 → 3
3+2 → 4 → 5-1 → 4

→ minus (1) because of peek ele has two times consider.

maxLen ⇒ 4 5 3 4

[Ans ⇒ n - maxLen ele's Need to Remove]

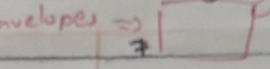
(26) LeetCode Q. No. (354) { Russian Doll Envelopes } → Try

Page No.: Date: YOUVA

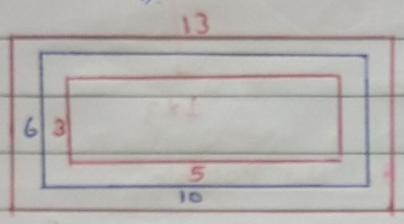
Ex: arr = { {3,5}, {7,13}, {6,10}, {8,4}, {6,7}, {7,11} }

w h

Envelopes =>



→ Ans => (3)



(a,b) (c,d)

=> (a,b) can get inside (c,d) if & only if.

a<c && b<d

Hints

① Sorting (Custom) [Comparable]

② LIS.

arr = { {3,5}, {7,13}, {6,10}, {8,4}, {6,7}, {7,11} }

Sort(row) = { {3,5}, {6,7}, {6,10}, {7,11}, {7,13}, {8,4} }

Sort(col) = { {8,4}, {3,5}, {6,7}, {6,10}, {7,11}, {7,13} }

arr = { {3,5}, {7,13}, {6,10}, {8,4}, {6,7}, {7,11} }

Sort(w) = { {3,5}, {6,7}, {6,10}, {7,11}, {7,13}, {8,4} }

Sort(w => asc, h => des) = { {3,5}, {6,10}, {6,7}, {7,13}, {7,11}, {8,4} }

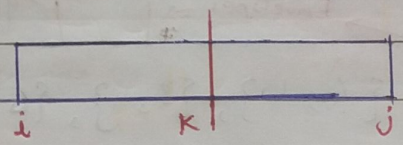
=> Code:-

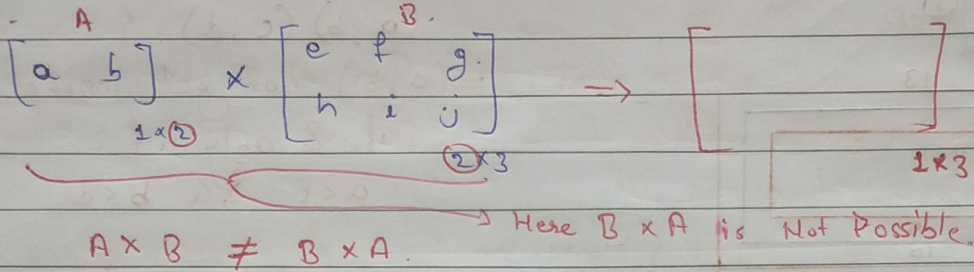
```
public class Envelop implements Comparable<Envelop>{
    int w;
    int h;
    Envelop(int w, int h){
        this.w = w;
        this.h = h;
    }
    public int compareTo(Envelop e){
        if(this.w == e.w)
            return e.h - this.h;
        return this.w - e.w;
    }
}
```

```
Envelop[] arr = new Envelop[n];
for(int i=0; i<n; i++){
    int w = envelopes[i][0];
    int h = envelopes[i][1];
    arr[i] = new Envelop(w,h);
}
Arrays.sort(arr);
```

Matrix Chain Multiplication

#Partition DP/MCM.



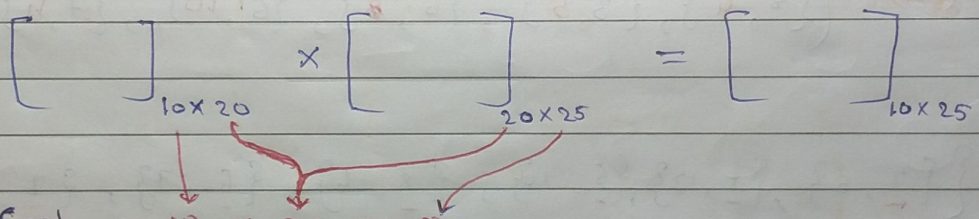


$A \times B \neq B \times A$.

A * B * C ⇒ (A * B) * C
↳ A × (B * C)

A × B × C × D ⇒ A(B(CD))
↳ A((BC)D)
↳ (AB)(CD)
↳ ((AB)C)D
↳ (A(BC))D

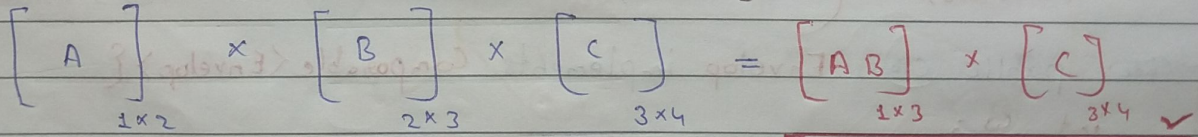
Cost of Multiplication



⇒ Cost ⇒ 10 × 20 × 25
↳ No. of Multiplication.

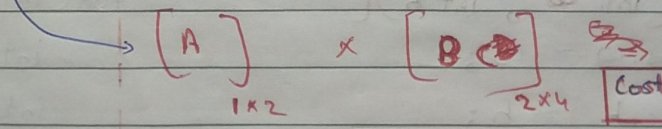
⇒ $A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5$ ⇒ $[\ ]_{m_1 \times n_5}$

#



Cost ⇒ 6 + 12 = (18) ✓

We Need to minimize the Costing.

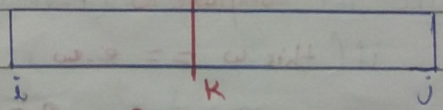


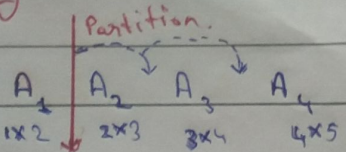
Cost ⇒ 24 + 8 = (32)

↳ $A_1 A_2 A_3 \ldots A_n$

→ Recursion

Try All Possible Combinations





cost(i,j) = cost(i,k) + cost(k+1,j) + C
A B

Input format ⇒ ⟹ 2-D arr ⇒

1	2	3	4
2	3	4	2

i k k+1 j

[(1×2)(2×3)] × [(3×4)(4×2)]

[1×3] × [3×2]

1 × 3 × 2 = x

1-D arr ⇒

1	2	3	4	2

i k k+1 j

4 Matrices

x = 1 × 3 × 2

arr[i] arr[k+1] arr[j+1]

2-D

	i=0	1	2	j=3
arr	1	2	3	4
	2	3	4	2
	A_1	A_2	A_3	A_4

partition → k

```
int Cost (int i , int j , arr) {
    if (i == j)  return 0 ;
    minCost = MaxValue ;
    for(int k = i ; k < j ; k++) {
        x = arr[i][0] * arr[j][1] * arr[k][1] ;
        tc = Cost (i , k) + Cost (k+1 , j) + x ;
        minCost = min (minCost , tc) ;
    }
    return minCost ;
}
```

arr[k+1][0] or

1-D

	i=0	1	2	3	j=4
arr	1	2	3	4	2
	A_1	A_2	A_3	A_4	

partition → k

```
for(int k = i ; k < j ; k++){
    x = arr[i] * arr[k+1] * arr[j+1] ;
    tc = cost (i , k) + cost (k+1 , j) + x ;
    minCost = min (minCost , tc) ;
}
return minCost.
```

Tabulation
- T.C ⇒ $O(n^3)$
- A.S ⇒ $O(n^2)$

Recursion + Memoization
- T.C = $O(n^3)$
- A.S ⇒ $O(n^2)$

Memoization :-

- i ⇒ n → 0
- j ⇒ m → 0.

→ Tabulation :-

- i ⇒ 0 to n
- j ⇒ 0 to m.

Memoization :-

- i ⇒ 0 to n.
- j ⇒ n to 0.

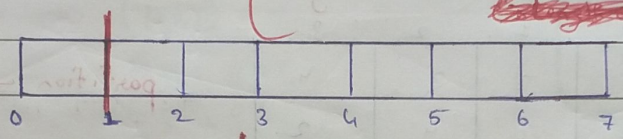
→ Tabulation :-

- i ⇒ n to 0
- j ⇒ 0 to n.

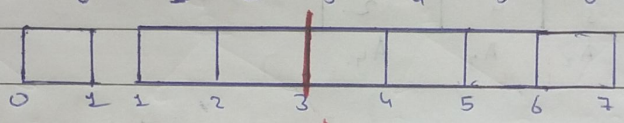
---

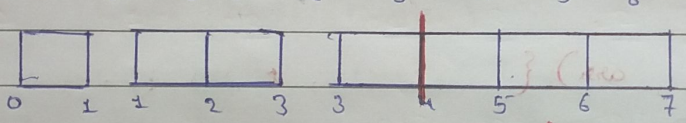
(27) Leetcode Q.No. (1547) { Minimum Cost to cut A stick. } → minimize

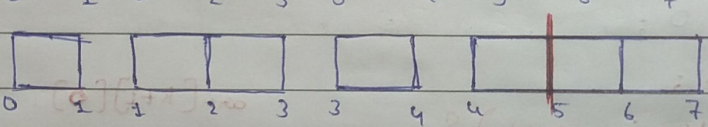
n = 7

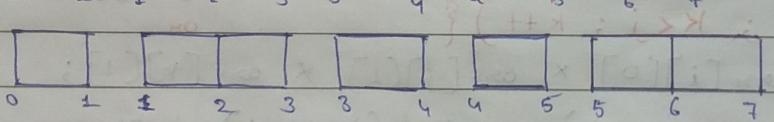


cuts = { 1, 3, 4, 5 }





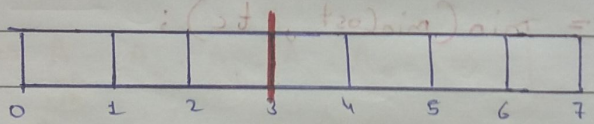




* Cost = 7 + 6 + 4 + 3

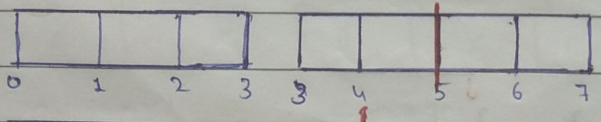
⇒ (22)

n = 7



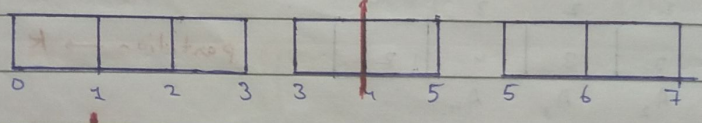
cuts = { 1, 3, 4, 5 }

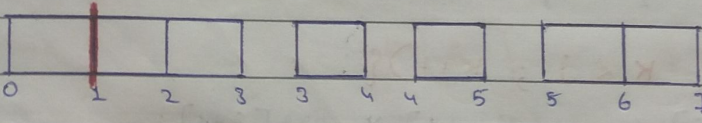
→ { 3, 5, 4, 1 }

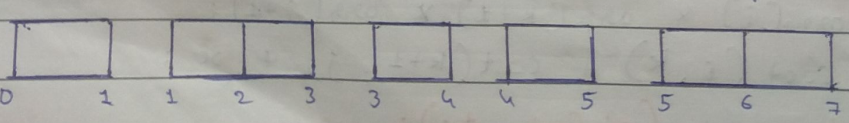


* Cost ⇒ 7 + 4 + 2 + 3

⇒ (16) → min.







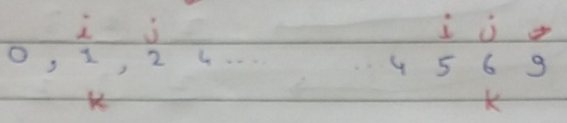
⇒ Cuts ⇒ Sort

⇒ cuts ⇒ front → 0 , end → n → Partition DP

n = 9 , cuts = {5, 6, 1, 4, 2}

arr = {0, 1, 2, 4, 5, 6, 9}

i k j



Cost (i, j) = cost (i, k-1) + cost (k+1, j) + len

len = arr[j+1] - arr[i-1]

if (i > j) return 0;

⇒ Code :-

```
for(int k = i ; k <= j ; k++){
    int len = arr[j+1] - arr[i-1] ;
    int tC = cost(i, k-1, arr) + cost(k+1, j, arr) + len;
    min = Math.min (min , tC) ;
}

return min;
```