1. Fibonacci
2. 0/1 KS - Unbound KS
3. LIS & Kadane's Algo
4. LCS & Pallindrome
5. MCM
6. DP on Tree
7. DP on Grid
8. others

Coin Change I & II

0/1 Knapsack & Unbound KS

↳ Subset Sum {1, 2, 3, 4, 11, 5} Target = 12 (11+1)

↳ Equal Sum Partition

↳ Count of Subset Sum

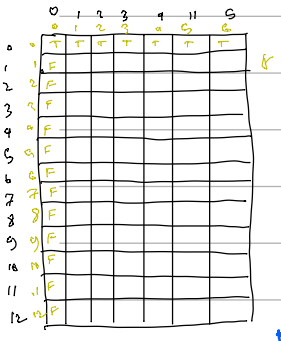
↳ Maximum Subset sum Difference

↳ Target Sum

↳ no. of Subset equal Given Difference.

take / not take : if > Target can't take

```
Solver( int i, int Target)
if (i==0 & Target==0) return True
if (Arr[i] > Target)
        return solver(i+1, Target)
else return solver(i+1, Target) |
       solver(i+1, Target - Arr[i])
```



if Arr[i-1] <= j

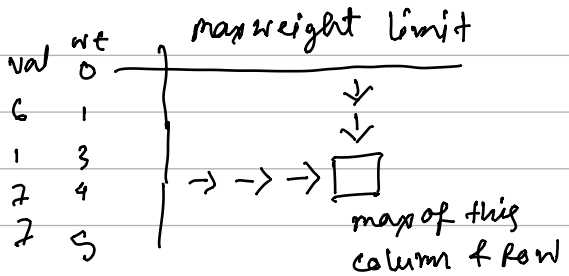
dp[i][j] = dp[i][j-arr[i-1]] or dp[i-1][j]

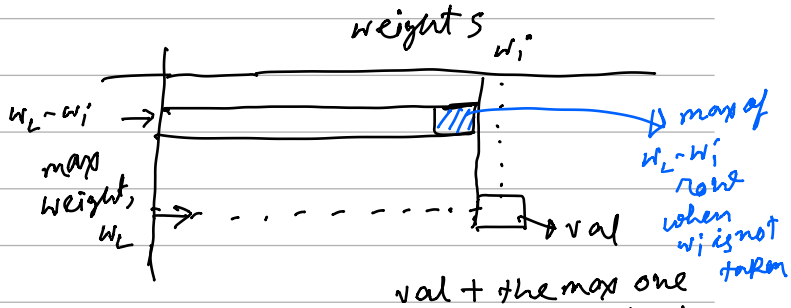
else dp[i][j] = dp[i-1][j]

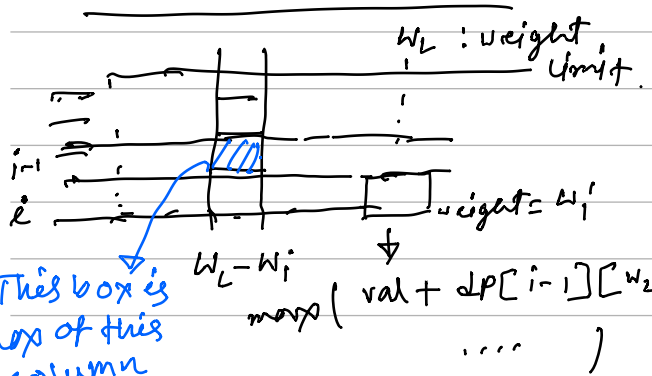
→ For unbound dp[i][j] = max(solver(i+1, target), solver(i+1, target - arr[i]), solver(i, target - arr[i]))

multiple instances are allowed

→ For bottom up 2 type of table possible







max(val + dp[i-1][$w_L - w_i$], ...)

Accordingly for unbound KS.

Print Knapsack :-

		X	0	1	2	3	4	5	6	7	8	9
revenue	cost	0	0	0	0	0	0	0	0	0	0	0
15	2	1	0	0	15	15	15	15	15	15	15	15
19	5	2	0	0	15	15	15	15	15	29	29	29
10	1	3	0	10	15	25	25	25	25	29	39	39
45	3	4	0	10	15	45	55	60	70	70	70	70
30	4	5	0	10	15	45	55	60	70	75	85	90

correct path

4, 3, 1, 5, 2

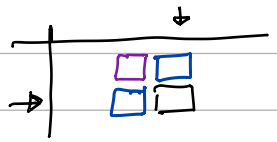
Longest Common Subseq (LCS)

- Longest Common Substring (LCST, Print LCST)
- SCSS, Print SCSS "abgh" "xygh" → "abxygh"
- Edit Distance "horse" → "ros" minimum no. of operation
- Largest Repeating Subsequence
- length of largest subseq in A which is substring in B.
- Subsequence pattern matching.
- Count how many time A appear in B.
  - Longest pallindromic Subsequence. LCS(A, Rev(A))
  - Largest Pallindromic Substring. LCST(A, Rev(A))
  - Count pallindromic Substring.
  - Min no. of Deletion / insertion to make A pallindrome

Subsequence

x b a b z h g b a b k z

LCS = b a b z



if A[i] == B[j]
dp[i][j] = 1 + dp[i-1][j-1]
else dp[i][j] = max(dp[i][j-1], dp[i-1][j])

Substring

x b a b z h g b a b k z

LCST = b a b

Simple; if (A[i] ≠ B[j]) dp[i][j] = 0
else dp[i][j] = 1 + dp[i-1][j-1]

at end search for the max.

→ Longest repeating Subsequence check if (A[i-1] == A[j-1] && i ≠ j)
— rest same as as LCS

⇨ Longest Increasing Sequence (LCS)

↳ LCS

↳ count no. of ways to find maximum in exactly K move.
(count no. of increasing subseq possible with K length).

○ print LIS: start from max go back and try all j where arr[j] < arr[i] & dp[j] = dp[i] - 1

∅ LIS in n log(n) using Patience Sorting.

⇨ Kadane's Algo :-

maximum Subarray Sum

eg: 9 3 -2 6 -19 7 -1 4 5 7 -10 2 9 -10 -19

iteration start → ..... another start →

	current best	overall best	
	9 → 9	9 → 9	# -ve from back don't take the train start your new train.
i = 1	7 → 9, 3	7 → 9, 3	
2	5 → 9, 3, -2		
3	11 → 9, 3, -2, 6	11 → 9, 3, -2, 6	
4	-3 : 9, 3, -2, 6, 19		
5	7 : 7		
6	6 : 7, -1		
7	10 : 7, -1, 9		
8	15 : 7, -1, 9, 5	15 : 7, -1, 9, 5	
9	22 : 7, -1, 9, 5, 7	22 : 7, -1, 9, 5, 7	
10	12 : 7, -1, 9, 5, 7, -10		
11	19 :		
12	23 :	23 : 7, -1, 9, 5, 7, -10, 2, 9	
13	13 :		

⇨ Matrix Chain Mul (MCM)

↳ MCM Print MCM

↳ Evaluate expression is True / Boolean Parenthesis

e.g (T or F) xor (F and F) make there partition such result = True ; return min no. of Partition Required.

↳ min / max value of an expression

↳ Pallindrome Partitioning.

↳ Scrambled String

str: abc → a cb , b ac → ca b , b ca

split str to a binary tree and swap children.

↳ egg dropping problem.

MCM,

0 1 2 3 4

10 , 40 , 20 , 30 , 50

```
for (K = i; K <= (j-1); K++)
    temp = solver(arr, i, K) + solver(arr, K+1, j)
         + (arr[i-1] * arr[K] * arr[j]);
```

10×30×50 + solver(i,K) + solver(K,j)

Dry run :- [2, 5, 7, 3, 1]

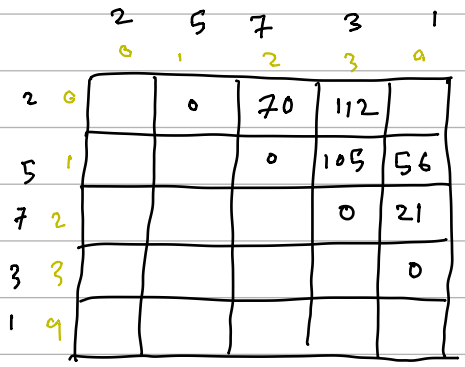
0 1 2 3 4

↳ 2×5 5×7 7×3 3×1

2×5 & 5×7 → 70

5×7 & 7×3 → 105

7×3 & 3×1 → 21



now,

2×5 & 5×7 & 7×3 → { 70 + 42 = 112 ; 105 + 30 = 135 }

5×7 & 7×3 & 3×1 → { 105 + 15 = 115 ; 21 + 35 = 56 }

O(n³) soln

Next, 0 1 1 2 2 3 3 4

2×5 & 5×7 & 7×3 & 3×1 → { i=0 K i+1 j-1 j=4

K=1 2×5×1 + dp(0,1) + dp(1,4) = 10 + 0 + 56 = 66

K=2 2×7×4 + dp(0,2) + dp(2,4) = 56 + 70 + 21 = 147

K=3 2×3×4 + dp(0,3) + dp(3,4) = 24 + 112 + 0 = 136

(2×40 40×2 2×40) 40×5

2 40 2 40 5

	0	1	2	3	4
2	0		160	320	
40			0	3200	560
2				0	400
40					0
5					

2×40 40×2 2×40

160 + 160

400 + 160

560

2×40 40×5

320 + 400

720

2×40 × 40×5

400

(2×40 40×2) (2×40 40×5)

160 400

2×40

2×2×5

20

0 1 2 3 4

2 40 2 40 5 8 9

0, 2, 0, 3, 1, 3, 1, 4

0 1 2 3

2 40 2 40

(0 — 2) (2 — 3)

2×40 40×2 2×40

.....

✓ # For boolean Parenthesis (count no. of unique Parenthesis s.t. evaluated exp^n is True)

exp: T & F | T ^ F

	T	&	F	|	T	^	F
	0	1	2	3	4	5	6
T 0	1	T	T&F	T&F|T	T&F|T^F		
& 1			F	F|T	F|T^F		
| 2				T	F		
T 3				1			
^ 4					F		
F 5					0		

True Count Table

	T	&	F	|	T	^	F
	0	1	2	3	4	5	6
T 0	0	T	T&F	T&F|T	T&F|T^F		
& 1			F	F|T	F|T^F		
| 2				T	F		
T 3				0			
^ 4					F		
F 5					1		

False Count Table

math ( ) op ( ) = Val

expression1 expression2

if op = & ⇒ exp1 & exp2 both need to be True then Val = True

⇒ exp1 or exp need to be False then Val = False

True count of (exp1 op exp2) = TrueCount(exp1) · TrueCount(exp2)

False Count of (exp1 op exp2) = Tc(exp1) · Fc(exp2)

+ Fc(exp1) · Tc(exp2)

+ Fc(exp1) · Fc(exp2)

Similarly for op = or and xor.

✓ Palindrome Partitioning

s: "a a b"

grid to tell whether s(i,j) palindrome

	a_0	a_0	b_1
a_0	✓	✓	✗
a_1	✗	✓	✗
b_2	✗	✗	✓

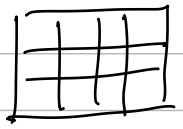
isPal

For bottom up:

$O(n^3)$ soln

a b b c a b

i j

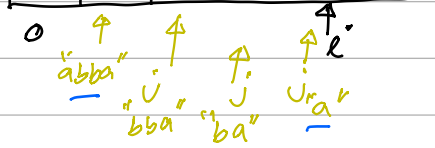


if (i,j) is pal dp[i][j] = 0

else 1 + min(ans, dp[i][k] + dp[k+1][j])

$O(n^2)$ soln

g a b b a c



if isPal[0][i] : dp[i] = 0

else check for(j = i ; j > 0 ; j--)

(if isPal[j][i]) dp[i] = dp[j-1] + 1

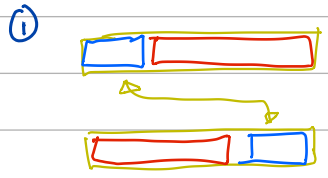
↳ Store the min of such.

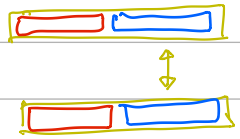
✓ Scrambled String –

a b a b g

q b a a b

① ② if when halfway



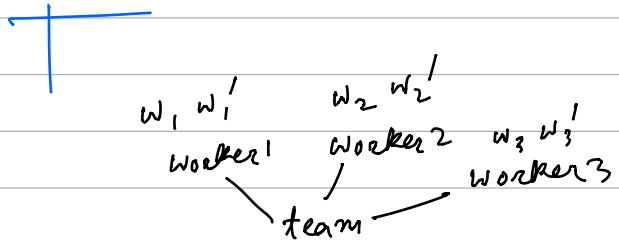


→ for this check ① as well also check this as well.

& can be scrambled of each other.

Red & Blue can be scrambled of each other to give a full scrambled.

✓ Egg Dropping :-

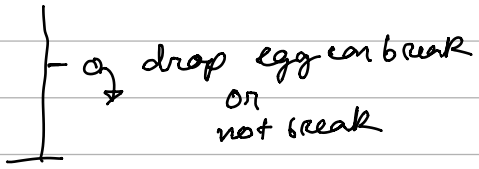


worker$_i$ can do a work in w_i or w_i' days (w_i can be $<$ or $>$ than w_i')

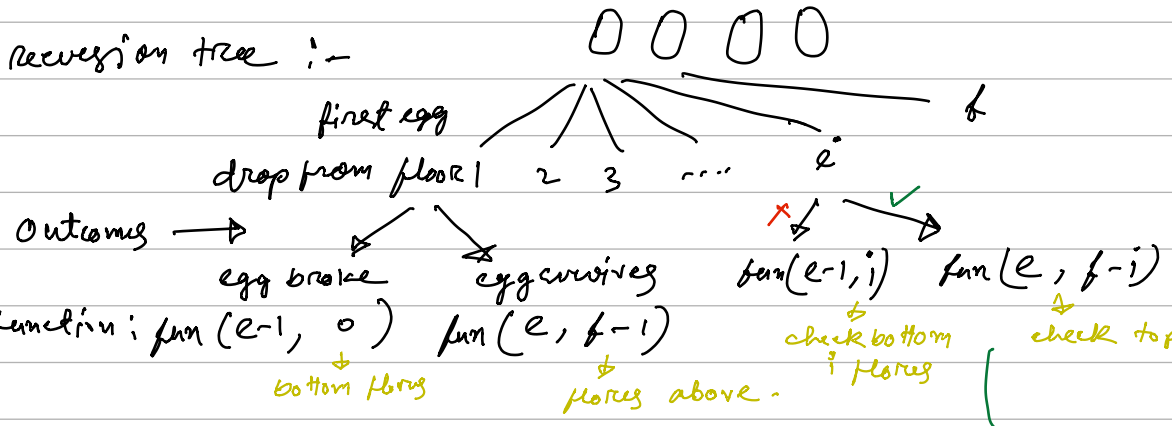
manager can gurantee a task in minimum how many days ?

$$\min\left(\max(w_1, w_1') + \max(w_2, w_2') + \max(w_3, w_3')\right)$$

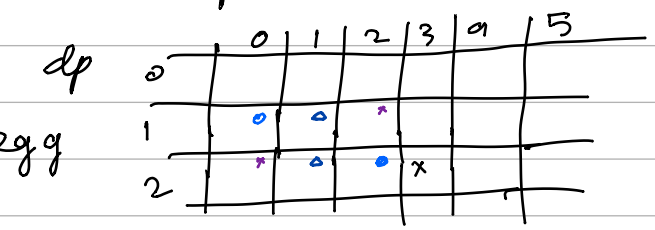
/ optimize the bad luck
/ Best of the worst times



Recursion tree :-



return best of each trial's worst.

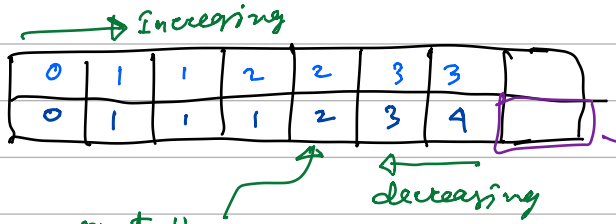


logic from here

$$dp[2][3] = \min\left(\max(dp[1][0], dp[2][2]),\ \max(dp[1][1], dp[2][1]),\ \max(dp[1][2], dp[2][0])\right)$$

check for each $i = 0$ to $f-1$

→ Further optimization



For finding dp of this box (i, j) we need to check $e-1$ $(0$ to $j-1)$ & e $(0$ to $j-1)$

we count the box where $[i][K] == [i-1][K]$ Ans

(use Bin Search)

more optimal approach to egg dropping problem.

e - egg count = 5 , f - floor count = 12

moves ↓ / egg →	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	2	3	3	3	3
3	0	3	6	7	7	7
4	0	4	10	14		

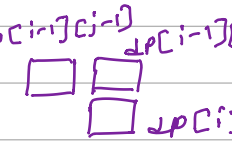
with 0 moves no floor can be checked

dp[i][j] represent using j egg with i no. of moves how many floors can be covered.

with 0 egg no. floor can be covered

with 1 egg in i move i floor can be checked.

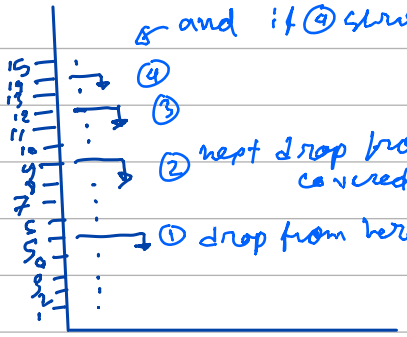
we found our required floor with 4 moves we could do that



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

* with j egg in i moves max floor possible
= 1 + in i-1 move using j eggs + in i-1 move using j-1 eggs. } i.e. **

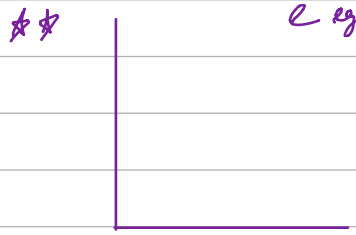
for example 5 moves 2 egg



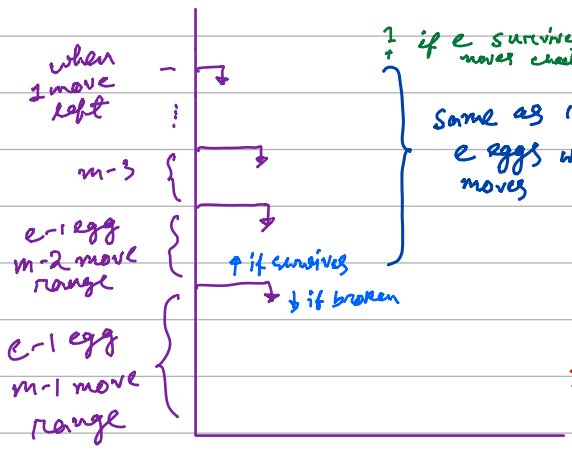
& and if ④ survives one more (last) move to check the above.

② next drop from here if breaks below undiscovered region can be covered by 1 egg in 3 moves .... e-1 egg m-2 moves

① drop from here if breaks below region can be covered with e-1 eggs and m-1 moves.



** e egg m moves



Important !!

#⭐, #Revision

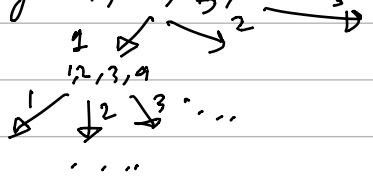
Digit DP

generate all numbers in b/w a range where sum is Target.

Target = 29 Range : [193 , 5522]

o/p :- 2999,
3998,
4997,

→ possibility 1, 2, 3, ... 9



Take a number & don't take & check whether < upperBound & sum == Target

dp [Target] [digit remaining]

→ hack : solver(upperlimit, Target, digit remaining) → generate all answer which are < 5522.

ans = solver(5522) − solver(193 − 1)

Bit DP

	c	p	j
0	1	1	
1	0	1	

abilities of person i → java, python,

j − java, c++