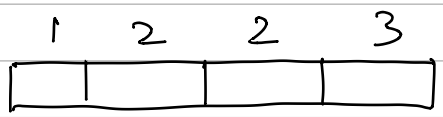
Decode Ways:

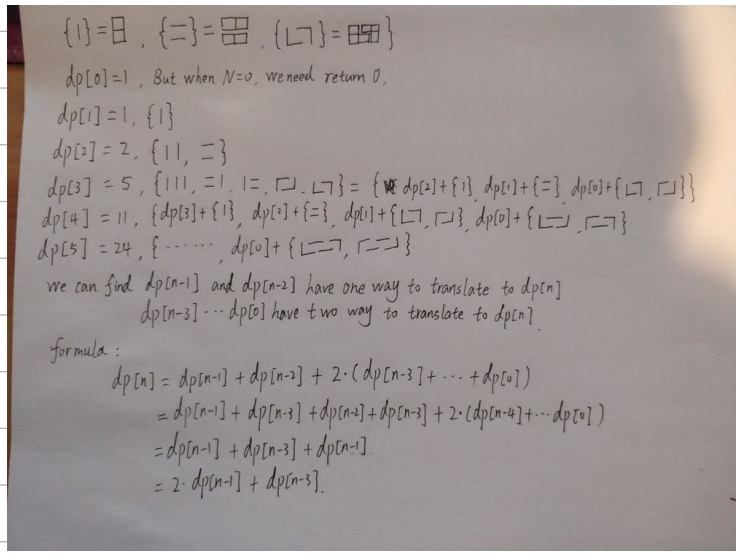
1223 → 1 2 2 3
12 2 3
1 22 3
1 2 23
12 23



2 {3}
1 {23}
2+1 {2 3}
23
3+2 {2 2 3}
22 3
2 23
{1 2 2 3
12 2 3
1 22 3
1 2 23
12 23}

dp[i] = dp[i+1] + dp[i+2]
+ other cases to consider.

Domino & Tromino Tiling:


{1}=▯, {=}=▯▯, {⌐}=▯▯▯
dp[0]=1, But when N=0, we need return 0.
dp[1]=1, {1}
dp[2]=2, {11, =}
dp[3]=5, {111, =1, 1=, ⌐⌐, ⌐¬} = {dp[2]+{1}, dp[1]+{=}, dp[0]+{⌐, ⌐}}
dp[4]=11, {dp[3]+{1}, dp[2]+{=}, dp[1]+{⌐, ⌐}, dp[0]+{⌐⌐, ¬⌐}}
dp[5]=24, {·····, dp[0]+{⌐⌐¬, ¬⌐⌐}}
we can find dp[n-1] and dp[n-2] have one way to translate to dp[n]
dp[n-3]···dp[0] have two way to translate to dp[n].
formula:
dp[n] = dp[n-1] + dp[n-2] + 2·(dp[n-3]+···+dp[0])
= dp[n-1] + dp[n-3] + dp[n-2] + dp[n-3] + 2·(dp[n-4]+···dp[0])
= dp[n-1] + dp[n-3] + dp[n-1]
= 2·dp[n-1] + dp[n-3].

Best Time to Buy and Sell Stocks I (normal max - min diff)
Best Time to Buy and Sell Stocks II (Can hold only one stock at a time; return Total maximum profit possible)
Best Time to Buy and Sell Stocks with Cooldown
Best Time to Buy and Sell Stocks with Transaction Fee
Best Time to Buy and Sell Stocks III (Total 2 Transaction allowed but one holding at a time)
Best Time to Buy and Sell Stocks IV (Total k Transactions allowed)

ϕ Buy & sell Stock with Transaction Fee :-

fee = 2 prices :	1	4	2	8	4	9
effective Profit :	-1	-1	-1 (1-2)	-1 (prev)	-1 vs (5-9) = ①	1
profit :	0	1	2-1-2 vs ①	(8-1-2)	(9+1-2)	9+1-2
		(4-1-2) = 1 (new)		= 5	= 3 vs ⑤	⑧ ✓

Max profit even after buying the i+1 day's stock

1 Holding : either continue previous holding or buy a new one spending previous profit.

No Holding : Either Not sell (continue previous) or sell using the holding at that day.

another Dry Run

	1	3	1	8	4	9
profit :	0	3+(-1) -2 = 0	1+(-1) -2 vs ⓪	8+(-1)-2 vs 0 = ⑤	4+(-1)-2 vs ⑤	9+(-1)-2 = ⑧
effective profit	-1	0-3 vs (-1)	-1	5-8 vs (-1) = -3	5-4 vs -1 = ①	8-9 vs ① = -1

profit = max(profit , price[i] + effectiveProfit - fee)

effective profit = max(effectiveprofit , profit - price[i])

ϕ Buy & Sell Stocks with Cooldown :-

	1	2	3	0	2
effective Prof :	-1	-1	* -3 vs (-1)	* 1-0 = ① vs -1	3-2 = 1
prof :	0	1 =	2	② vs 0+(-3) = -3	2+1 = ③ vs 2

* effective Prof = max(effecive Profit , profit[i-2] - prices[i])

profit[i] = max(profit[i-1] , effectiveProfit + prices[i])

	1	2	3	5	2	9
effective Prof :	-1	-1	* -3 vs (-1)	1-5 = -4 vs (-1)	2-2 = ⓪ vs -1	4-9 = -5 vs ⓪
prof :	0	1 =	2	5+(-1) vs 2 = ④	2+-1 vs ④ = 1	9+0 vs 9 = ⑨

Buy & Sell Stock III : Hard

2 Non overlaping Transaction ;

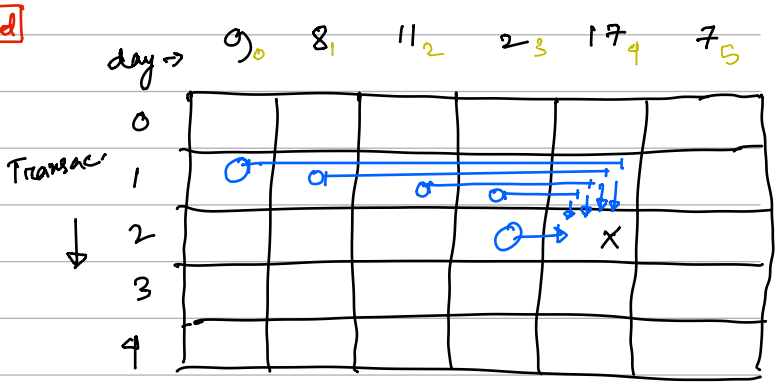
2 DP forward & Backward.

Forward[i] : maxprofit if Buy & Sell happend on or before ith day

Backword[i] : " " " " " " " " after ith day

Buy and Sell Stock IV :- Hard

K Transactions allowed.

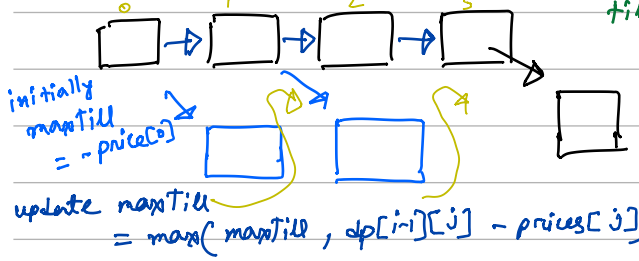


dp[i][j]
= maximum profit at jth day with having i transactions.

$dp[i][j] = \max\left(\underbrace{dp[i][j-1]}_{\text{no transaction, just take max profit of previous day}} \text{ or } \underbrace{dp[i-1][k] + (price[j] - price[k])}_{\text{take } k^{th} \text{ days } i-1 \text{ transaction } \& \text{ trade; sell } k^{th} \text{ day's stock at today's price}} \right)$

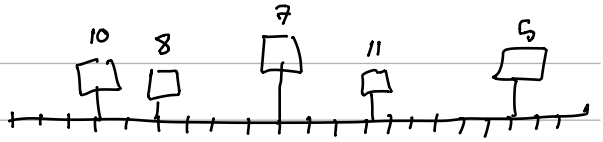
Optimization :

instead. store the maximum $(dp_k - BuyPrice_k)$ till j



compare max $\begin{cases} dp_1 + (SellPrice - BuyPrice) \\ dp_2 + (SellPrice - BuyPrice) \\ dp_3 \\ \vdots \end{cases}$

Highway Billboard Problem :-

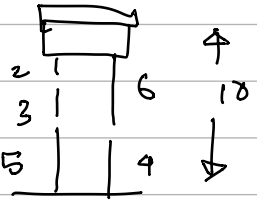


Road of length m Km. each billboard atleast 1 Km apart.
billboards array size n cannot take profit from 2 billboard in 5 Km vicinity.
calculate max profit.

$O(n^2)$ → LIS variant.
$O(m)$ → store the max at each Km progress throughout the road.

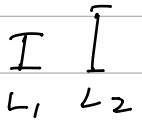
Tallest Billboard :- Hard

[1, 2, 3, 4, 5, 6, 4]



return Tallest billboard possible.

solver(int idx, vector<int> rods, int diff)



if rods[idx] is added in L_1 diff + rods[idx]
" " " " L_2 diff − rods[idx]

	1	2	3	4	5	6	4
must in L_1	1,0	3,0					
" " L_2	0,1	0,					
nowhere	0,0						

6 | 3 | 2 | 1 | 1

Similar Qn

partition array so that difference of Sum is minimum. Hard

2 −1 8 4 −2 −9 → Total Sum : 2

Target Sum = 1
find the sum of any 3 elements Sum closest to 2.

	0	2	-1	4	-2	-9
0	0	0	0	0	0	0
1	0	0	0			

Subset Sum / Knapsack will give TLE
use Meet in the Middle algorithm.

※ Not a DP problem.

∅ check : Closest Subsequence Sum.

Partition Array for maximum Sum :-

1, 15, 7, 9, 2, 5, 10

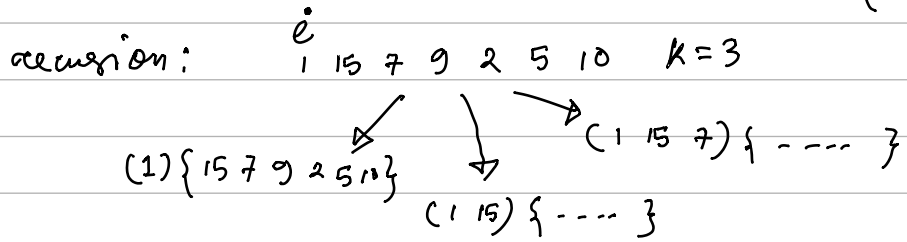
15, 15, 15, 9, 10, 10, 10

k = 3

at max subarray len = k

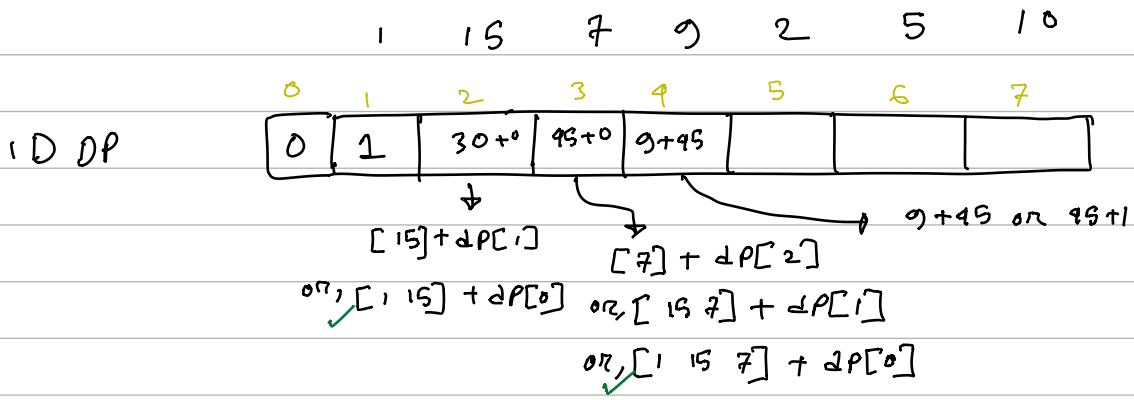
subarray elements will become max(subarray)

recursion:



take 1 len subarray from ith pos.
or 2 len subarr.
or 3 ..
:

bottom up:



Minimize the difference between Target & chosen Elements :-

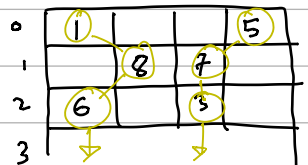
1	2	3
4	5	6
7	8	9

Target = 13

ans = 0

Approach 1 : DP

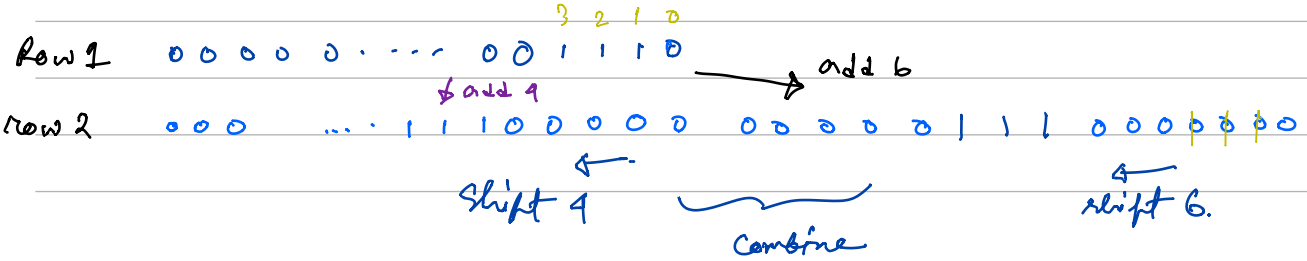
solver(int row, int sum)



call for dp[3][15] should no do the work again memoize.

#⭐, #Revision, #Trick, #Tricky business

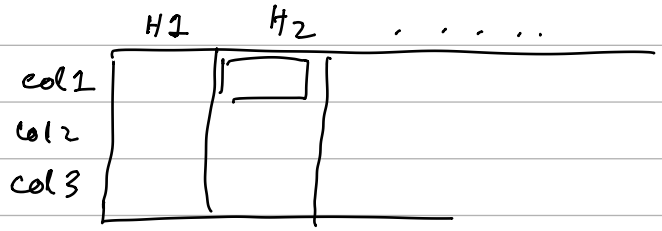
more Interesting Sol^n:
using BitSet;



Paint House I

House	col1 cost	col2 cost	col3 cost
1			
2	:	:	:
3	:	:	:
9			

No 2 adjacent houses have same color.
return minimum cost.



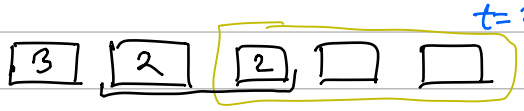
DP → To paint house h 2 and using col 1 wat's the min cost.

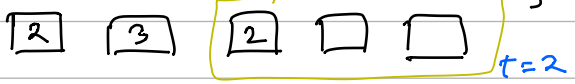
ans: cost to paint col1 + min(H_1 col2 , H_1 col3) # since H_1 can't have col 1.

Paint House III :- Hard

Same as before, color - cost
- This time some houses may be already painted.
- Constraint on same color neighborhood.

crux : 3D DP dP[m][target+1][n+1]

1) 

2) 

} these 2 subproblems are different because number of neighborhood is not changing in case 1 but changing in case 2.

2 Key Key board :-

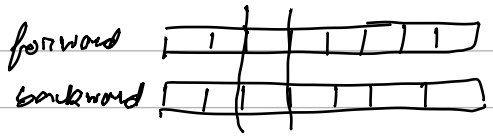
"A" → cP → A
cPP → AA
cPcP → AA

given n return min no. of c and/or p required.

⇒ if n = prime ans = n
else : ----- } has to do with multiples.

→ while(n >= i)
if (n%i == 0) pf += i ; n /= i ; } code to generate sum of prime factors.
else i++;

- LIS
- Print LIS
- Longest Bitonic Subsequence

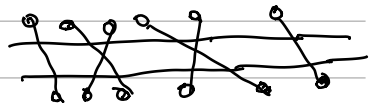


if(for[i] > 1 & back[i] > 1) ↳ then LongestBitonic = for[i] + bac[i] - 1;

ensuring atleast 1 length increasing ; ensuring atleast 1 length decreasing

Totally Increasing / decreasing is not Bitonic.

- Maximum Non Overlapping bridges

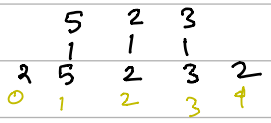


LIS on north bank one by one on the basis of its other end at South Bank.

- LeetCode Version : Uncrossed Lines ✓

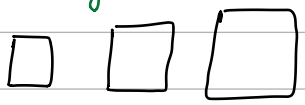
find LIS in the array : [1, 4, 2, 0, 3]

5 to 5 ; 2 to 2 ; 3 to 3



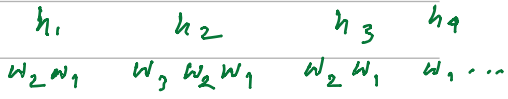
Trick is to sort in Reverse order of each element's multiple Occurence

Then apply n log(n) LIS

- Russian doll envelopes :  width & height.

tell how many stack of box inside box possible.

same here → sort height →



Sort width in reverse then n log(n) LIS.

—— LIS end ——

next page ...

Perfect Squares :-

$n = 11$ ans = $2^2 + 2^2 + 1^2 + 1^2 + 1^2$

or $3^2 + 1^2 + 1^2$ ✓✓

dp :

0	1	2	3	4	5	6	7	8	9	10	11	12
0	1											

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

dp[4-1] +dp[1]

or

dp[9-4] + dp[4] +1

$n = 13 \rightarrow$ dp[13-1] + dp[1]

dp[13-4] + dp[4] = ✓

dp[13-9] + dp[9] = ✓

Catlan Numbers :-

https://en.wikipedia.org/

$C_4 = C_3C_0 + C_2C_1 + C_1C_2 + C_0C_3$

0	1	2	3	4	5
1	1	2	5	14	42

dp[5] = dp[0] × dp[4] + dp[4] × dp[0]
+dp[1] × dp[3] + dp[3] × dp[1]
+dp[2][2] ✗ 1 time not 2 time.

$C_3 = C_2C_0 + C_1C_1 + C_0C_2$

$= 2 + 1 + 2 = 5$

dp[5] = (14+5) × 2 + 4

= 42

```
for( int i = 2 ; i<=n ; i++)
    sum = 0
    for(j = 0 ; j < i ; j++)
        sum += ( dp[j] × dp[i-j-1] )
```

combination of Balanced Paranthesis :-

n Pairs	count	list
0	1	""
1	1	()
2	2	(()), ()()
3	2×1 + 1×2 + 1×1 = 5	

for 3 : ( ) ↓ n=2 ↓ n=0 ; ( ) ↓ n=1 ↓ n=1 ; ( ) ↓ n=0 ↓ n=2

Put elements of

ans: ((())) ""

(()()) ""

(())()

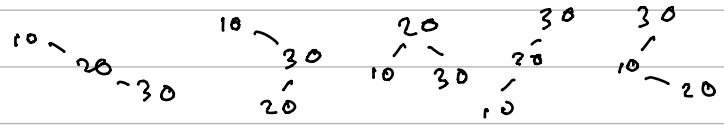
("") (())

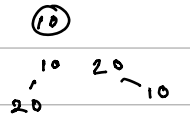
("") ()()

Count no. of possible BSTs given count of nodes:

10 20 30

node Count	no. of BST
0	1
1	1
2	2
3	2×1 + 1×2 + 1×1 = 5



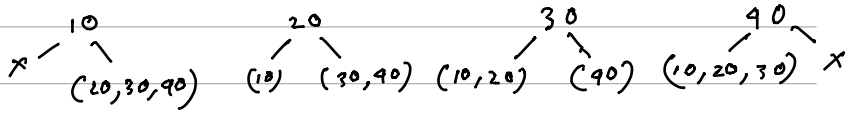


Root is 10: count(20, 30)

Root is 20: c(10) c(10)

Root is 30: c(10, 20)

4 — 5×1 + 2×1 + 1×2 + 1×5 = 14



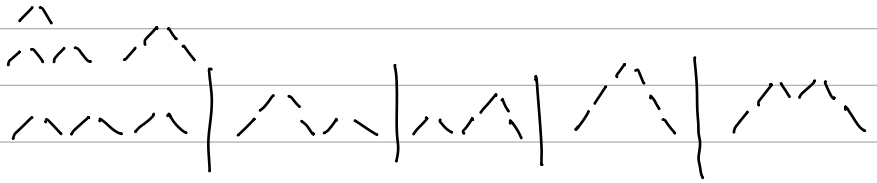
1 × f(3) + f(1) × f(2) + f(2) × f(1) + f(3) × 1

Count Valleys & mountains :-

constraints : at no time upstroke & downstroke.

n = 1

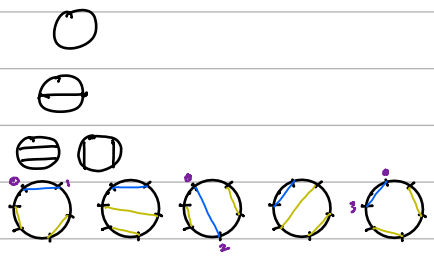
n = 2

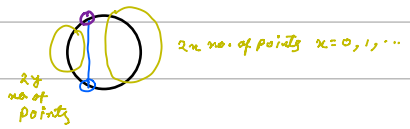


for n = 3 — count = 5

Non Intersecting Chords in Circle :-

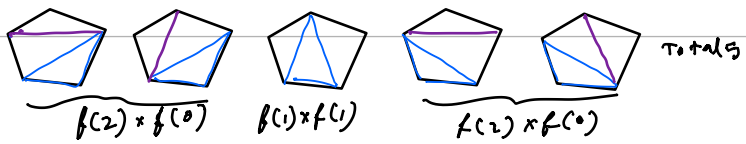
Points		
0	n = 0	1
2	n = 1	1
4	n = 2	2
6	n = 3	5



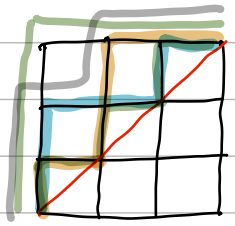


Ways of Polygon Triangulation :-

(sides) n	Nos	ways
2	0	1
3	1	1
4	2	
5	3	



- Maximum Score of Triangulation
- Dyck Words (VVVHHH, VHVVHH ...
  no. of H <= no. of V at any point.
-



go from bottom left to top right such that never goes below bottom.

n=3 ans=5

Matrix Chain Multiplication :-

Burst Balloons :

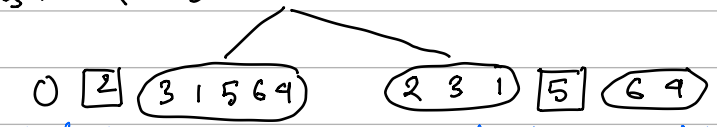
balloons : a b c d e

if d is popped coins = e x d x e

if e is popped coins = d x e x 1 (end is 1)

E.g. balloons : 2 3 1 5 6 9

Tree diagram :



Last balloon popped is 2

ans = 0 + solver(3 1 5 6 9) + 2 x 1 x 1

Last balloon popped is 5.

ans = solver(2 3 1) + solver(6 9) + 5 x 1 x 1

→ Last balloon so only edge balloons of cost 1 remaining

8 [3 4 6 5 2 2] 9 3

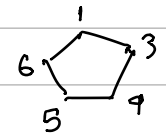
To solve this sub problem

0 + (4 6 5 2 2) + 3 x 8 x 9 or (3) + (6 5 2 2) + 4 x 8 x 9 or (3 4) + (5 2 2) + 6 x 8 x 9 or ...

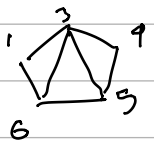
take the maximum coins ...

Minimum Score Triangulation of Polygon :-

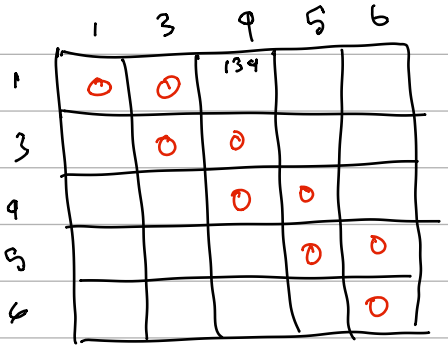
[1 3 4 5 6]



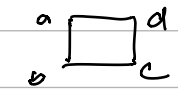
→



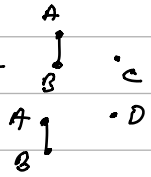
cost = 3×5×2 + 1×3×6 + 3×4×5.



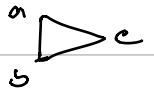
[x], [x y] → have cost = 0.



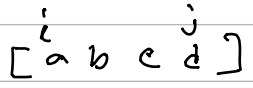
→ Base BC



abc + bcd OR abd + bcd

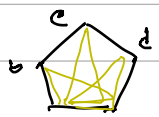


→ [a b c] →



[a b c d e]

abd + (ab) + (bcd)
acd + (abc) + (cd)



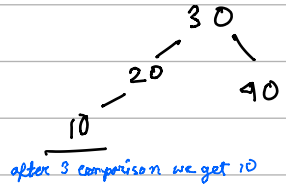
abe + (ab) + (bcde)
ace + (abc) + (cde)
ade + (abcd) + (cde)

Boolean Parenthesization

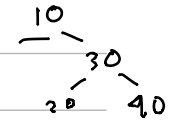
1) Min no. of Parenthesis req to make it True
2) Count no. of ways it can give True

Optimal Binary Search Tree :

nodes : [10 20 30 40]
no. of Searches : [8 9 3 5]



after 3 comparison we get 10



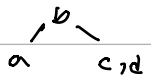
after 1 comparison.

Total Cost = 8×3

8×1

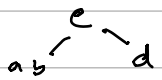
nodes a b c d

↓ root



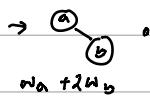
or a b c d

↓ root

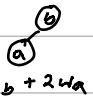


	a	b	c	d
a	w_a			
b	x	w_b		
c	x	x	w_c	
d	x	x	x	w_d

a b →

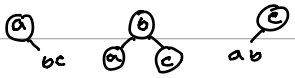


or



$w_a + 2w_b$ $w_b + 2w_a$

abc →



$w_a + 2dp(bc)$ $w_b + 2w_a + 2w_c$ $w_c + 2dp(ab)$ ✗

bcd → cost $w_b + 2w_c + 2w_d$

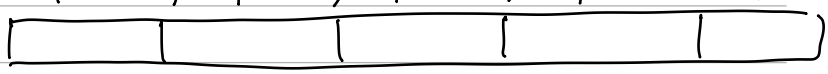
abcd → cost $w_a + 2w_b + 3w_c + 4w_d$ ✓

Not $w_a + 2w_b + 4w_c + 4w_d$. ✗

Rod Cutting :-

① gfg given the selling price of a particular length of piece of Rod (Not MCM ... 1D DP ...)

len, price 1|100 , 2|120 , 3|800 , 4|850 , 5|860

dP: 

dP[3] = 900 (just sell 3 length in 800 Rs & 2 length in 100 Rs.)
4|850

② LeetCode given positions to be cut broken into, cost = length of the sub piece before breaking. (MCM)

Palindrome Partitioning :-

Palindrome Partitioning I : Return all pallindrome Substring
Palindrome Partitioning II : Minimum no. of cuts so that cuts are Pal
Palindrome Partitioning III : Minimum no. of character changes required to make K substring from it .. all are pallind.
Palindrome Partitioning IV : Return true if it's possible to make 3 non empty pallindrome.

↳ dp memoize isPallindrome.

```
dP [ ][ ]      bool isPal(int i, int j, string & s)
                  if(i >= j) return true;
                  if(dp[i][j] ≠ -1) return dP[i][j]
                  if(s[i] == s[j]) return dP[i][i] = isPal(i+1, j-1, s)
                  return dp[i][j] = false
```

✓ For PP II first needed 2D MCM To get if S[i..j] is Palindrome or not then → 2D MCM Method to find minimum cuts checking all substr.
↳ 1D DP to check min cut from 0 to i and hence 0 to n.

Longest Common Subsequence

Longest Pallindromic Subsequence (MCM way or LCS(A, rev(A)))

pending ⊗ Count pallindromic Subsequences of length 5 : (Leet code) Hard

Count Pallindromic Subsequence (classic 1 MEM DP)

String: a b c c b e

L=1 & L=2 Trivial

abce = abc + bce − be

a	b	c	e	cc
a	b	c	e	cc
a	b	c	e	cc
b	c	e	cc	
c	e	cc		
e	cc			
cc				

a, b, c, e, cc

	a	b	c	c	b	e
a	1	2				
b	x	1	2			
c	x	x	1	3		
c	x	x	x	1	2	
b	x	x	x	x	1	2
e	x	x	x	x	x	1

bccb

b	bb
c	bcb
c	bccb
b	
cc	

bcc + cc b + 1

b	bcb
c	bcb
c	b
cc	bccb

bb = 1

Total = 9

Count Pallindromic Substrings: (1 MEM) Boolean DP

Just Count the no. of True.

Longest Common Pallindromic Substring: Boolean DP & another DP

if (S[i] == S[j] && dpIsPal[i+1][j-1]) ans = max(ans, dp[i+1][j-1] + 1)

Count Distinct Subsequences :-

s = "abcbac"

""	a	b	c	b	a	c
count 1	2	4	8	8×2 − 2 = 14	14×2 − 3 = 27	27×2 − 8 = 50
	""	"" "b"	␣ c	␣ b Repeat	a	c
	"a"	"a" "ab"	a ac	a ab	Repeat	ac
			b bc	b bb		bc
			ab abc	c cb		abc
				ab abb		Repeat
				ac acb		
				bc bcb		
				abc abcb		

Those are same occurrences of Previous b

LeetCode Distinct Subsequences I (no. of subSEQs matching target)
Distinct Subsequences II (Previous Q. Count no. of Subsequences in itself)

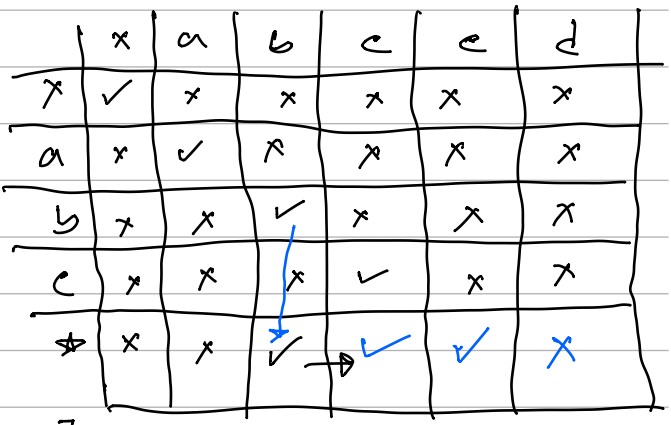
Count Related : 1) Count Pallindromic Sequences (also substrings)
2) Count Distinct Pallindromic Sequences (also substrings)
3) Count Pallindromic sequence of length 5.
4) Count SubSEQs matching Target.

1) ✓
3) X
4) Same as count occurances of a string T in S as subString / subSequence.

2) → Hard AF https://youtu.be/fvYlinirmFg?si=RnCDgQ318kEjY2Zc

- LCS
- Count longest Repeating Subsequence ✓ (LCS(A,A) condition $i \neq j$ && A[i] == A[j])
- wildCard Matching : '*' can become any seq "abc", "", "dg"...
- Regular Expression Matching :- "s*" → "", "s", "ss" sss..
  "a*" → "", "a", "aa" ....

a b] c c d
a b] c *
check
a b
a b c *
a b] c
a b] c *
a b c] c
a b c] * c



a b c c c] c
a b c *

a b c c c d
a b c * x

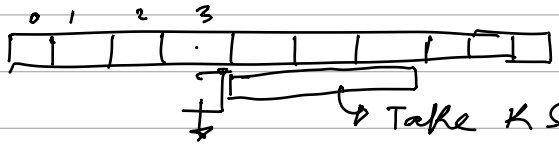
- Edit Distance :
- Scrambled String ; 3D DP
- minimum ASCII Delete Sum (LCS won't work coz ASCII value)
- Minimum Cost to make 2 strings identical (Total − LCS)

Kadane's Algo:

if sm + num[i] >= num[i] take the family train
sm + num[i] < num[i] start your own train.

- Max Subarray Sum with K times repeating Array.
- Maximum Subarray Sum with atleast Size K:

Trie:

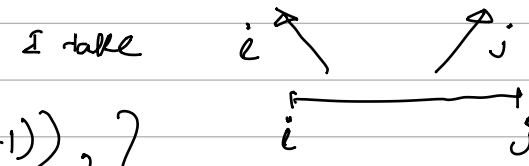


- Maximum difference b/w no. of zeros & Ones in Substring :-

1 1 0 0 0 1 0 0 0 1 0 1 1 1 1
-1 -1 1 1 1 -1 1 1 1 -1 1 -1 -1 -1 -1 → apply Kadane

- Egg Dropping Problem:
- Optimal game strategy :- arr: [10 20 30 10 50]

[Do best expect worst]



$i + \min((i+2, j), (i+1, j-1))$,
$j + \min((i+1, j-1), (i, j-2))$ } → I chose the best among these

	1	5	233	7
1	1	5		
5		5	233	
233			233	233
7				7

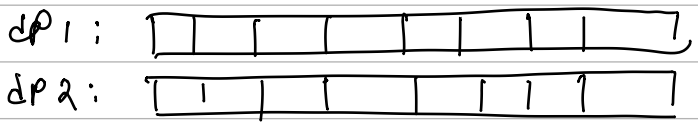
1 5 233

- Knight's Probability in Chessboard.
- Distinct Transformations :-

a a a b b c c d → a b c d

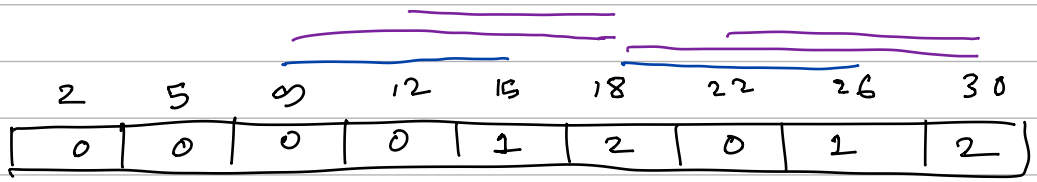
no. of ways to delete elements to change into target.

- Max of 2 non overlapping Subarray Sum :-

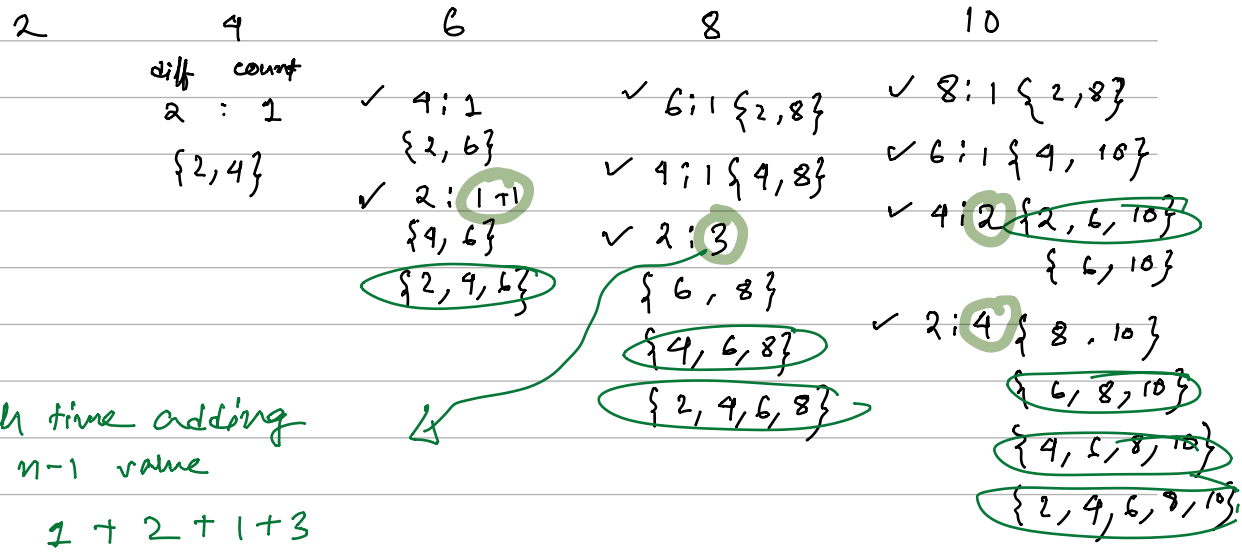


ans = max(dp1[i] + dp2[i])

- Max of 3 non overlaping subarray sums :-
- Max of K non Overlaping Subarray Sums :-
- Arithmatic Slice I :- count no. of subarrays in AP



- Arithmatic Slice II : Count no. of subsequences in A.P.



Word Break :-

words : [" leet", "code", "leetcod", "pep", "coding", "pepcoding"]

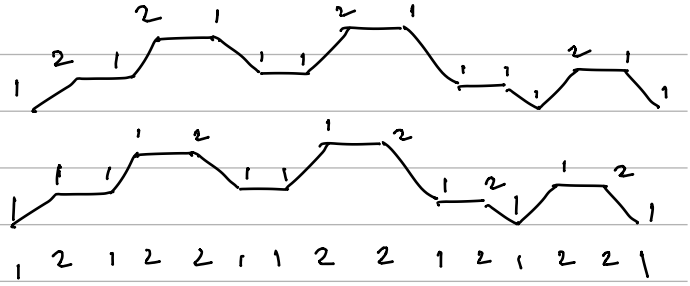
string : "leetcodepepcoding"

↳ return how many valid sentences are possible

L e e t c o d e p e p c o d i n g
0 0 0 1 0 0 0 2 0 0 3 0 0 0 0 0 4
ans

Trie or set can be used.

Temple Offerings :



temple at higher altitude must get more offering than it's neighbor lower temple.

max → 1 2 1 2 2 1 1 2 2 1 2 1 2 2 1

→ sum up.