Dec-7th

DP - 1

$dp(i, x) = x \longrightarrow$ value of DP.

(i: states of dp.)

for harder problems, key is getting the states.

Partition problem: n can be written as a sum in how many ways?

$4 = 1+1+1+1$
$= 1+1+2$
$= 1+3$
$= 2+2$
$= 4$

$dp(i, taken)$ → by taking an extra state we are fixing the limit on permutations!!

# of ways to enumerate i only using nos $\geq$ taken.

$\therefore dp(i, taken) = \sum dp(i, j)$

(i: 0-N, j: taken 0-N.)

$\therefore$ avg no. of transitions = $O(N)$

$\Rightarrow$ each state requires $O(N)$ steps for calculation

$\Rightarrow$ Total complexity = $O(\#\text{states} * \#\text{avg. transitions})$

$\Rightarrow$ In this case $O(N^2 \times N) = O(N^3)$

$\therefore$ Complexity depends on both states & transitions

$\longrightarrow$ make this as low as possible for the same q/n.

$dp(i, taken) = dp(i-1, taken) + dp(i, taken+1)$

here only $O(N^2)$.

2) Max. Sum Subarray:

1) Kadane's algo.

2) dp algo.

$dp[i] \rightarrow$ max sum subarray ending at index i.

$\therefore dp[i] = max(arr[i], arr[i] + dp[i-1]);$

$\therefore$ Max. Sum Subarray$_{opt} = \max_{i=0 \text{ to } N}$ (of all $dp[i]$)

$O(N)$

3) Knapsack: Weight W Bag. N items. each item p_i, w_i

Find max. profit possible s.t $\sum w_i \leq W$.

$N \leq 5000$

$W \leq 5000$

Ans) $dp(i, w)$ → max profit possible by taking first i items with weight $\leq w$.

0/1 : Version:

$$\therefore dp(i,w) = \max \begin{cases} dp(i-1, w-w_i) + p_i, \\ dp(i-1, w) \end{cases}$$

Complexity = $O(N \times W)$

If we are allowed to take any no of items:

$$dp(i,w) = \max \begin{cases} dp(i-1, w-w_i) + p_i \\ dp(i-1, w-2w_i) + 2*p_i \\ \vdots \\ dp(i-1, w) + n*p_i \\ dp(i-1, w) \end{cases}$$

↓ more no. of transitions per state

But this is equivalent to,

$$dp(i,w) = \max \begin{cases} dp(i, w-w_i) + p_i \\ dp(i-1, w) \end{cases}$$

→ this takes into accnt all the transitions.

↓ Consider first i infinite supply.

Now even this has $O(N \times W)$!!

Find ans = $dp(N, W)$

4) Longest increasing Sub-seq:

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

$dp[i]$ = length of LIS ending at i

$\therefore$ Ans = $\max_{i=1}^{N} (dp[i])$

$$dp[i] = \max_{1 \le j < i \,\&\, a[j] < a[i]} (1 + dp[j])$$

$\therefore$ Time complexity = $O(N^2)$.

But can we do better?

O(NlogN) Sol^n:

Pruning the # of values stored in our dp.

#obs 1: if the last value is same, we only keep the pair with larger length.

$dp(i) = \max_{j<i,\ a[j]<a[i]} (dp(j)+1)$

1	6	4	5	6
1	2	2	3	4

Now note we have 2 6's

~~(6,2)~~ & (6,4)

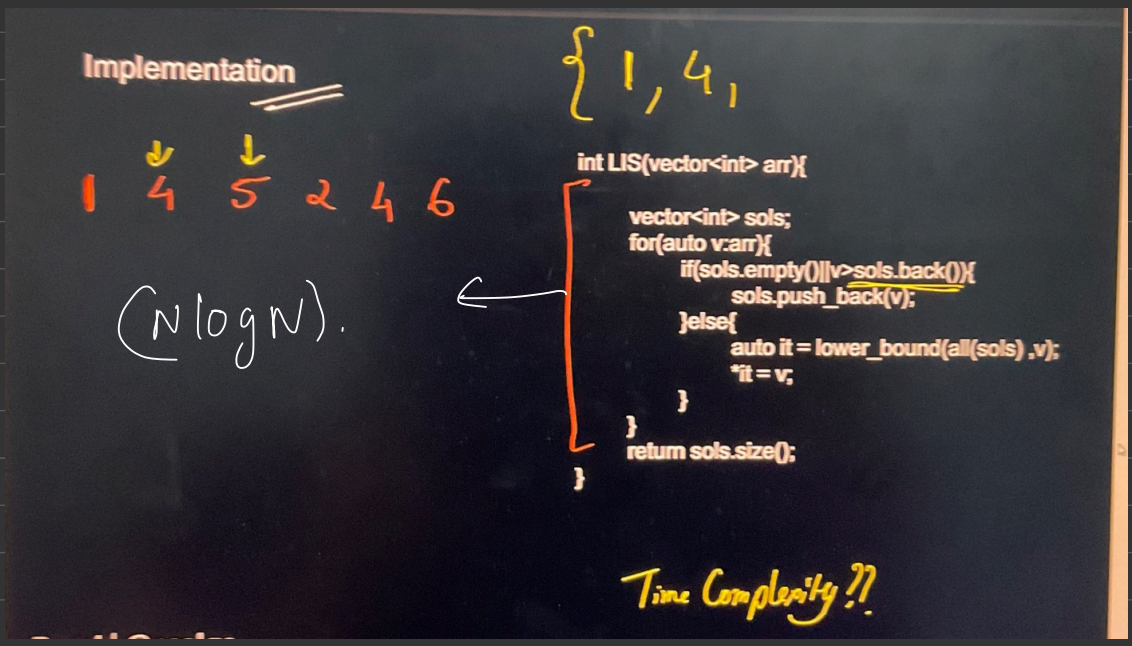
We basically dont need (6,2) as if an element greater than 6 comes we wld append it only to the rightmost 6 right!!

#obs 2: if length is same, we only keep the smallest value.

Similar explanation.

"

these for as we progress, we have (value, length) both increasing together and this remains an "invariant".



1 2 3
1 4 5 2 4 6. now this 2 has length 2

1 4 5 2

1 2 3 2

1 1 1

1 2 5 4

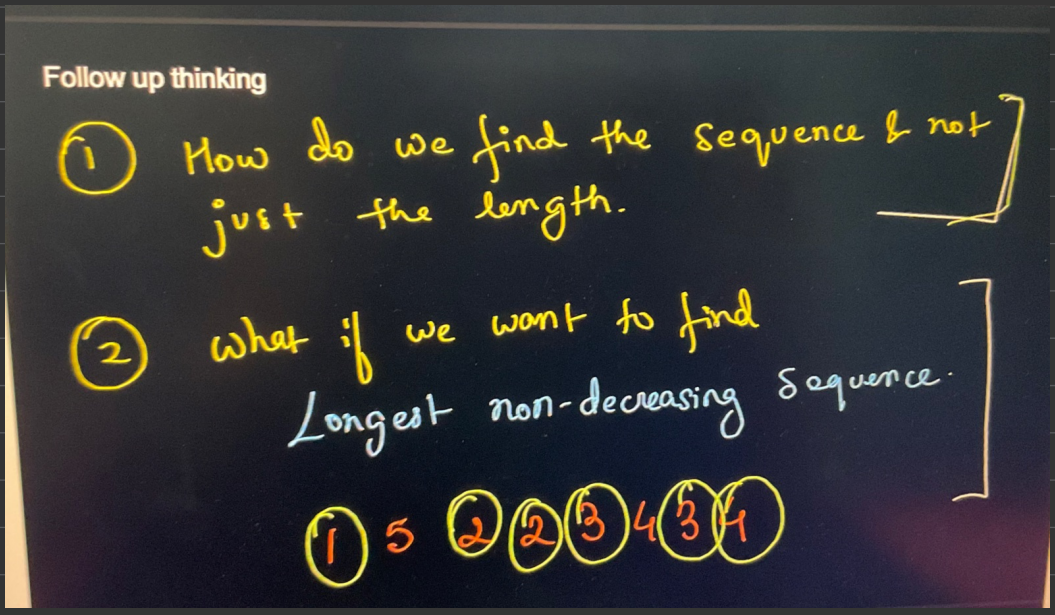
1 2 3 2

1 1 1

1 4 5 6 → final.

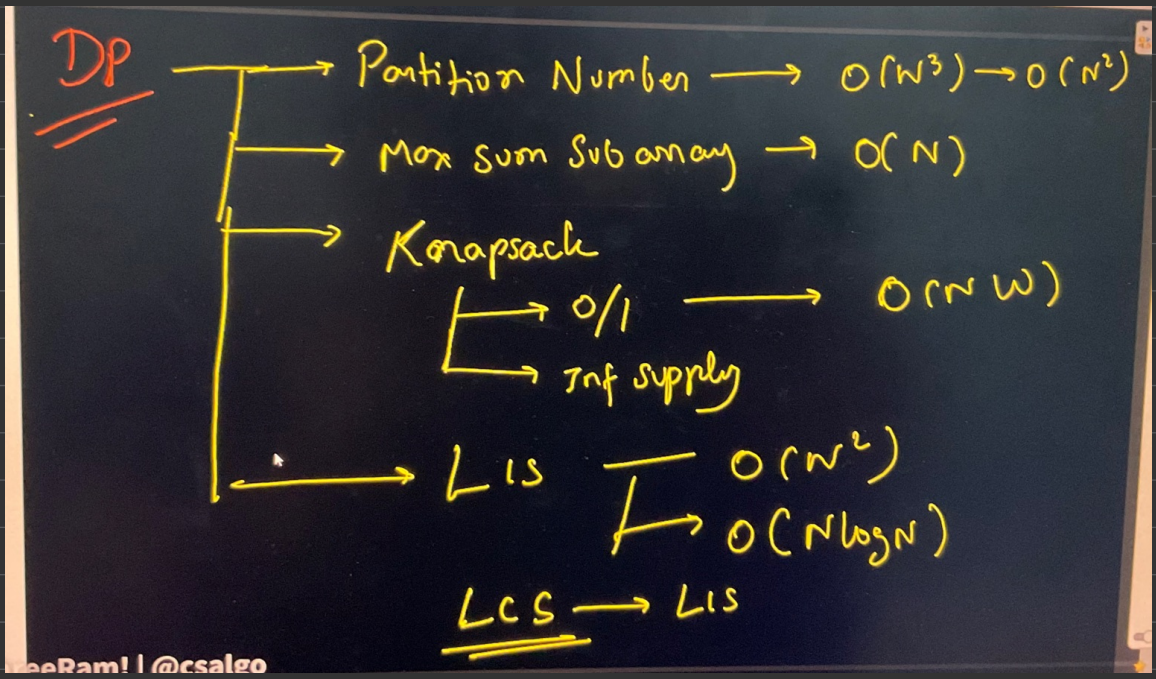
1 2 3 4

∴ final length of sub would be our ans.



Always for permutations, we solve them using a "limit" state so that we don't enumerate extra combinations.

upto now :



6th May

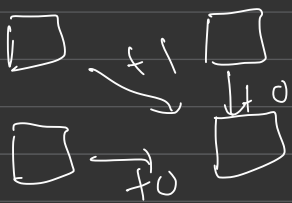
LCS: → if one has all distinct then can be reduced to LIS but Cant if not.

dp[i,j] = LCS of $S_1[0 \cdots i]$ & $S_2[0 \cdots j]$

i,j match → 1 + dp[i-1, j-1]

i,j dont match → max(dp[i-1, j], dp[i, j-1])

TC = O(N × M).



backdp[i,j] = (i-1, j-1, 'd')

Deque q/n:

Deque

P_1, P_2 both play optimally. Find max diff $SP_1 - SP_2$.

$dp(l,r)$ = a player's turn now and gives us the max diff he can create if both play optimally using $[l,r]$ indices.

***

$$\therefore dp[l,r] = \max\begin{cases} arr[l] - dp[l+1, r] \\ arr[r] - dp[l, r-1] \end{cases}$$

we take $a[l]$, then other players perf. diff score is $dp[l+1, r]$

$\therefore$ we need to substract it.

→ P2score

$P_1S - P_2S = opt$

$P_1S + P_2S = sum\ of\ arr.$

1 to 999 10

960

(Qn) n players $\xrightarrow{i^{th}\ \text{player}}$ (x_i, y_i) $a+b \le n$.

↓

either take him / his x_i / his y_i

Exactly 'a' players for x_i, 'b' for y_i

Score = sum of attributes → max. this score.

$$dp[i][a][b] = \max \begin{cases} dp[i-1][a-1][b] + x_i[i] \\ dp[i-1][a][b-1] + y_i[i] \\ dp[i-1][a][b] \end{cases}$$

$O(N^3)$.

↓

can do a state rd^n sol^n once check

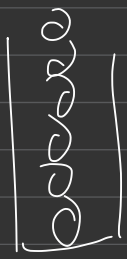
Live-3: dp1.

dp + Sorting.

(Qn) game theory

↳ Shld be combinatory

↳ Shld be played with same strategy. (impartial).



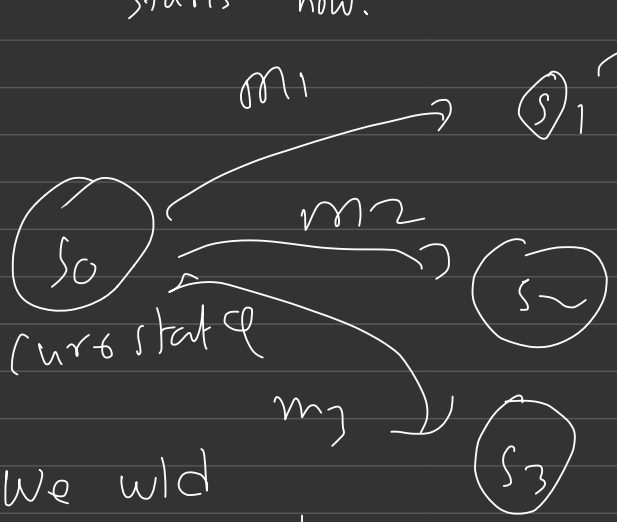
N rocks. m guns.

$K_1, K_2, \ldots K_m$ bullets.

if both players play optimally who will win if P_1 starts.

dp(#no. of stones left) = 0/1 for the player that starts now.

transitive:



if $dp(S_1) = 1$ we wld not want to go there.

∴ we push to the state s.t. $dp(S_i) = 0$

$$\therefore \boxed{dp(i) = 1 - \prod_{j=1}^{m} dp(i - K[j])} \quad **.$$

if all 1 then only dp(i) = 0.

∴ time complexity : O(N×M).

Practice q/ns

q3) N pple, M lines of code

vector <int> Ai

↓

errors / line of code by i^{th} person.

# of distinct ways $(X_1, X_2, \ldots X_n)$

↓ lines of code / person.

s.t. Program has $\leq B$ bugs

Print ans % (10^9+7).

dp[i][j][B] $\equiv$ # ways of writing j lines using i pple having $\leq B$ bugs.

↓ persons, ↓ lines, ↓ bugs.

We need dp[N][M][B].

$dp[i][j][B] \equiv dp[i-1][j][B]$

$+ dp[i][j-1][B-a[i]]$ *

how see, ur states must be s.t. everything equates.

dp[i][j][B] (i^th person may/may not write)*

↓

using i persons, # of ways to write exactly 'j' lines of code with exactly 'B' amount of errors.

Now mem limit is O(NxMxB)

Can we do better?

Yes blud.

we only need one prev. state for calculating current state.

∴ dp[2][500][500] is enough.

also. p = i & 1 ⟶ 1 if i is odd.

p = i & 1 ⟶ 0 if i is even.

p^1 ⟶ ~negation of that.

Bruv never use ~ operator as

we dont know how many bits are stored. just use XOR with 1.

***

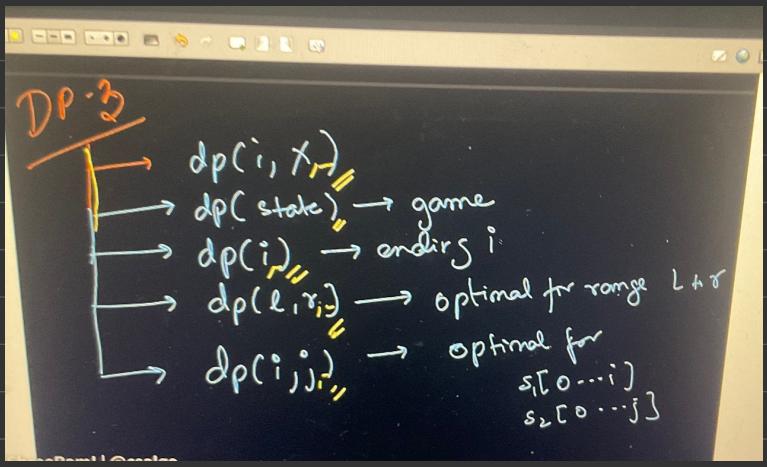
Now be very careful regarding base cases.

1) Hard code the base case & dont use it again in the for loop blud.

Start from the next thing.

L-R DP (DP-2)

9- May



Q1) Array a, n. $a_1, a_2, \ldots a_n$

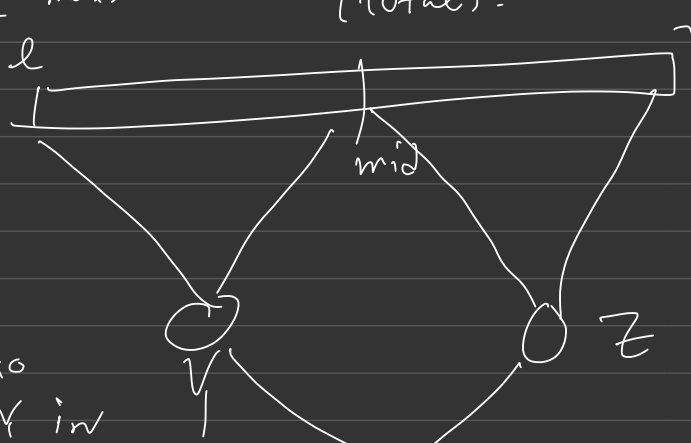
$N \leq 500$
$a_i < 100$.

operation: a_i a_{i+1}

↓ merge → reward = $a_i * a_{i+1}$.

$(a_i + a_{i+1}) \% 100$

∴ we perform merges until single element is reached
find max. reward. (total).



We need to store Y in our dp state if diff permutations may lead to different Y's.

else only $dp(l,r)$ is sufficient.

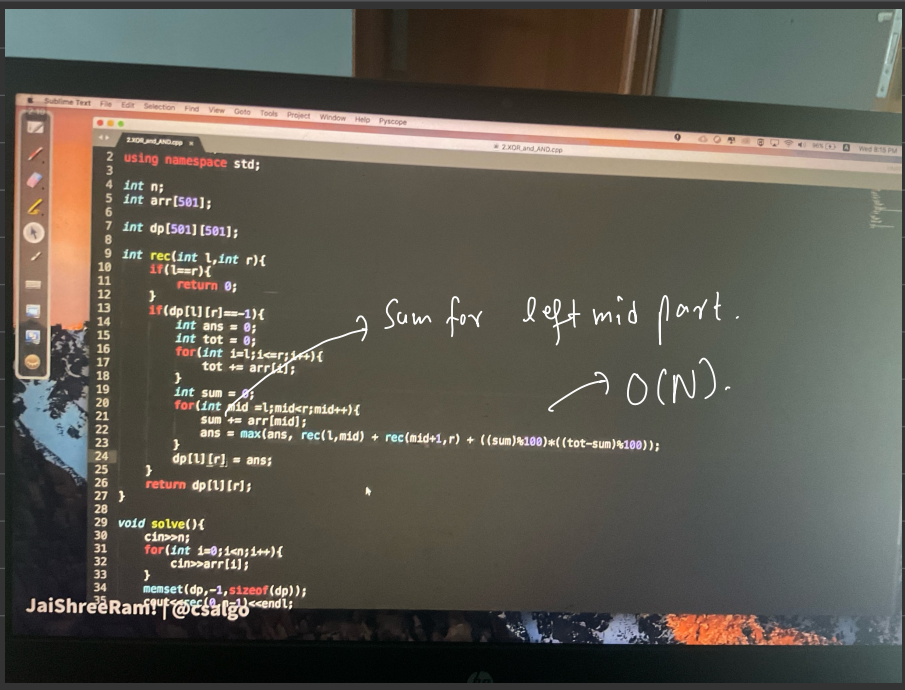
here irrespective of how we merge, finally for a range $a_1, a_2, a_3 \ldots a_n$ we get $(a_1+a_2+\ldots a_n)\%100$ as our final val.

$$\therefore dp(l,r) = \max_{\text{all } mid=l}^{r-1} \left(dp(l,mid) + dp(mid+1,r) + \left(\sum_{l}^{mid} a\right) * \left(\sum_{mid+1}^{r} a\right) \right)$$

we iterate over all mids to get max profit.

$O(N^3)$

by prefixsum



Q2) $N \le 50$, $x, y, z, a_i \le 50$.

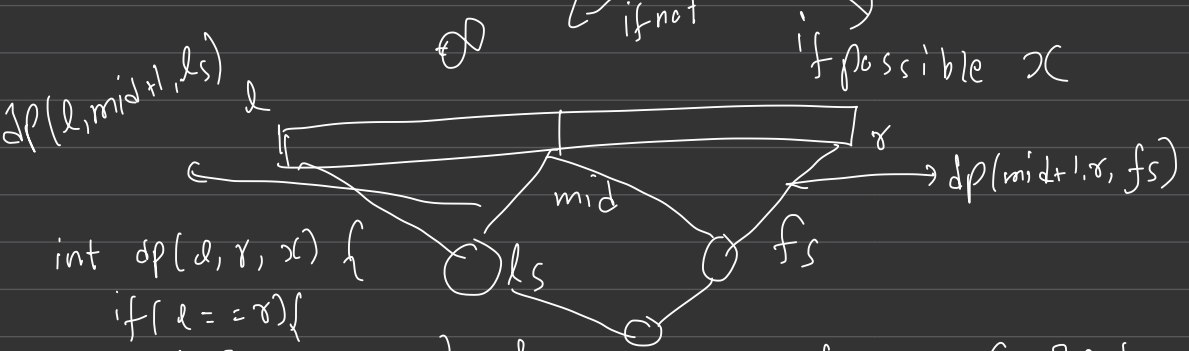
$a_1, a_2 - - - -, a_N$

$a_i \quad a_{i+1} \longrightarrow (a_i * a_{i+1})$

$(a_i * x + a_{i+1} * y + z) \% 50$.

Now we cant use $dp(l, r)$.

We need $dp(l, r, \text{merge\_val})$ → merge val 0-50 possible write.




```
int dp(l, r, x) {
  if(l==r){
    if(arr[l]==x) return 0;
    else return 1e9
  }
  if(done[l][r]){
    return memo[l][r][x]
  }
  for(i=0; i<50; i++) memo[l][r][i] = 1e9;
  for(mid=l; mid<r; mid++){
    for(fs=0; fs<50; fs++){
      for(ls=0; ls<50; ls++){
        memo[l][r][(fs*a + ls*b + c)%50] = min(dp(mid+1, r, fs) + dp(l, mid, ls) + fs*ls);
```

∴ for one $[l, r]$ pair we get all possible x values in one go.

$\therefore O(N^2 \times N^3) = O(N^5)$

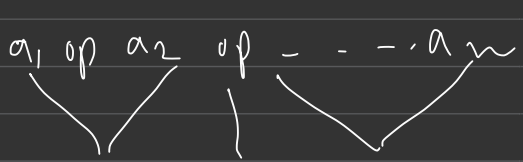
Q3) Put brackets to get max. possible value.

$3+2-5*3$

max dp(l, r, val)

Sometimes we might not only need the max value / what the qn asks us. But also some other auxilary values to reach our final answer.

here we also have to keep min dp(l, r, val).



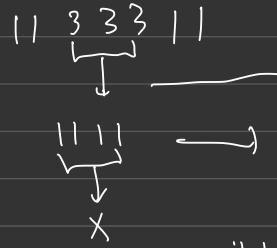
∴ for x-y to be max, we need max x - min y.

∴ we take all 4 combo's to ensure we get the max value.

∴ We need to calc both min & max using only min&max of smaller sub problems at each step.

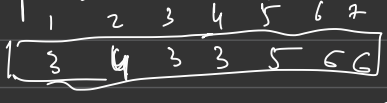
(Q4) N a_1 a_2 a_3 ... a_N

remove a subarray with all identical elements and get reward of (#elements in subarray)^2,



Find max. reward possible by removing all array elements.

Now this dp state is actually really tough.



Now consider first 3. it has → 3 choices either removed by itself. or removed with left 3 or removed with right 3 or removed with all 3's together.

(case i) $1^2 + dp(2,7)$

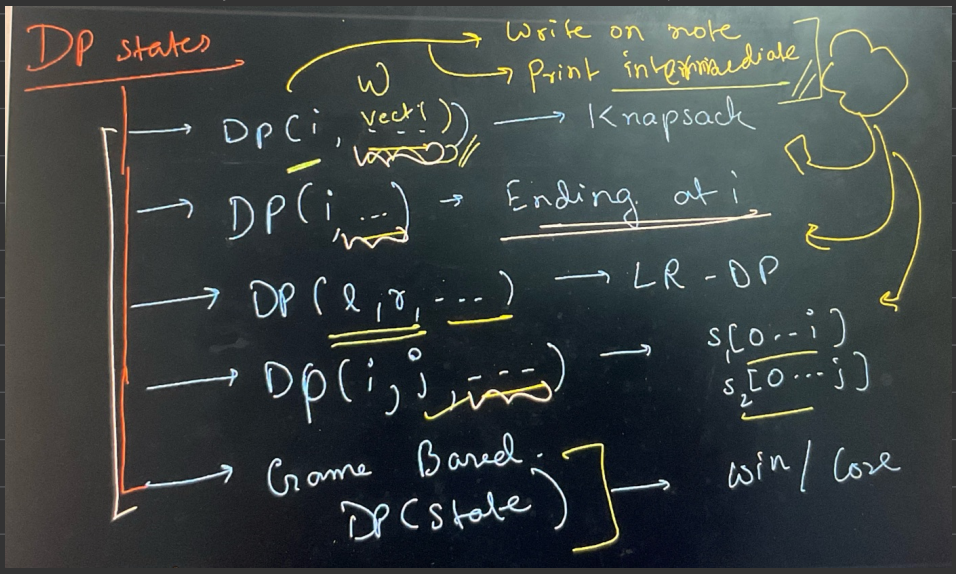
(case iii) $dp(2,3) + dp(4,7,1)$ ← 1 to indicate one left out 3.

$dp(l, r, K)$ → K is the no. of elements in the starting of the array.

```
int rec(int l, r, k) {
    if(l>r) return 0;
    if(dp[l][r][k]==-1){
        int ans = (k+1)*(k+1) + rec(l+1, r, 0);
        for(i=l; i<=r; i++){
            if(arr[i]==arr[l]){
                ans = max(ans, rec(l+1, i-1, 0) + rec(i, r, k+1));
            }
        }
        dp[l][r][k]=ans
    }
    return dp[l][r][k];
}
```

MIXED DP

first think of which style of DP.



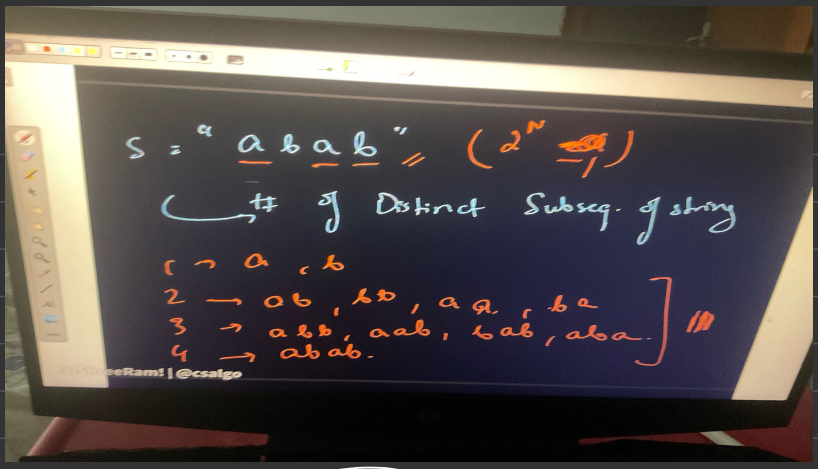
(Q2) No. of valid combinations to fill the 2 paranthesis so that these balanced 2

((? ?) ? ?)

(( )) ( )

(((( ))))

Q3)



a b a

start with empty { }

{ } → { } + {b} (append b)

{a} → copy {a} {ab}

{ } {b} + {a} {ba} → copy

{a} {ab} + {aa} {aba}

here {a} shld be removed.

∴ We need to double prev state and substract our last occured position

∴ We start with empty string dp[0] = 1

dp(i) = 2 * dp(i-1) - dp[last occurence of i - 1]

↓ maintain an array.

O(N)

Q4) Seq. Counting.

Find No. of seq. of length n st no two neighbouring pos. have 1. using 0, 1, 2.

dp[0][i] → # of valid seq of length i starting with 0.

dp[0][i] = dp[1][i-1] + dp[2][i-1] + dp[0][i-1].

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

dp[2][i] = dp[1][i-1] + dp[2][i-1] + dp[0][i-1].

∴ O(N*3)*3 ≈ O(N).

Q4). N, K (0, 1) No no K consecutive pos is same.

dp[pos, last char, last char Count] → # of possible valid seq.

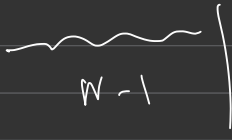
↓ 0-N ↓ 2 ↓

Q5) N presses allowed.

4 keys possible are ① A, ② ctr-A, ③ ctr-C, ④ ctr-V

Find max no. of A's that can be printed.

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



②, ③ not at all optimal.

∴ we shld do either ① or ④.

Now consider doing ② at any time in the middle no pt if ③ not done.

and if ②, ③ are done it is always optimal to perform ④ over ① as atleast 1 a is printed.

$$\therefore dp[i] = \max \begin{cases} dp[i-1] + 1 \\ dp[i-3] * 2 \\ dp[i-4] * 3 \\ \vdots \\ dp[i-(i-1)] * i \end{cases}$$

$O(N^2)$.

Digit - D.P

13-May

3 more state ideas.

② Digit D.P. → easy

③ Tree D.P. ⟶ Very tough

① Bitmask D.P. → easier.

↓

Very common.

Q1) Find the number of integers in $[L, R]$.

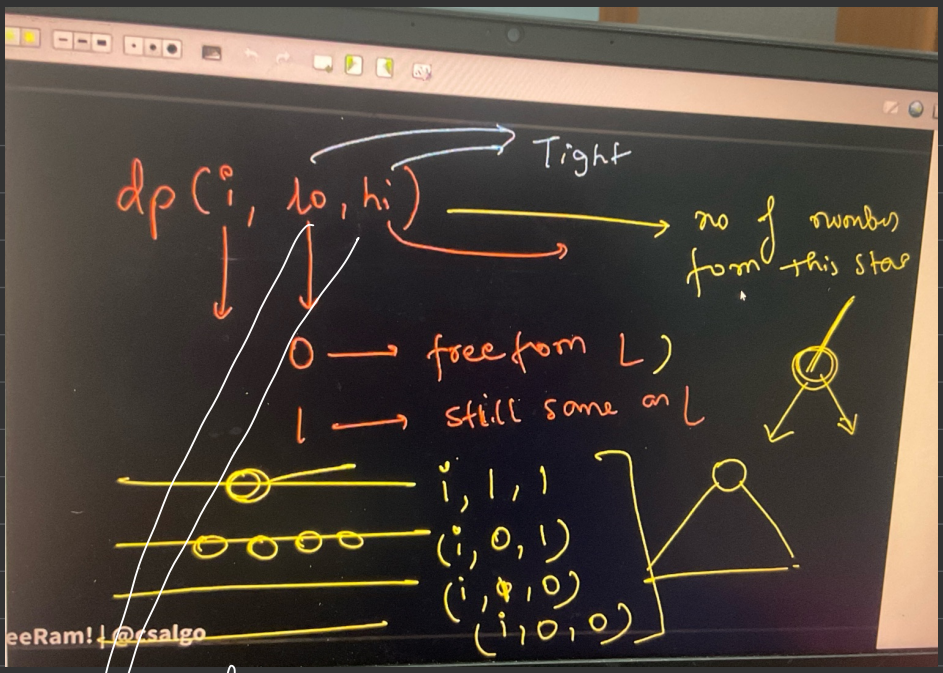
↓

understanding digit dp using this qn.

$O(1) - (R - L + 1)$

→ Once we are free from L, R any node in the same level will have equal no. of answers. So calculating for 1 internal node is enough.

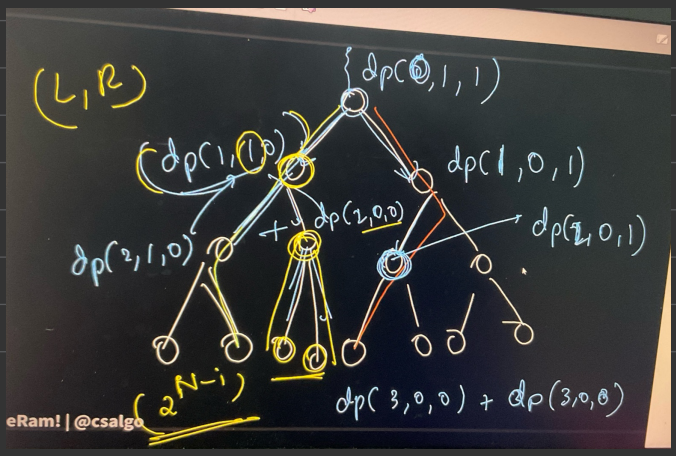
→ Same L, then freedom



only 4 types of dp.

these 2 to restrict our dirn.

(L, R) → are fixed. once we are free
#nos = 2^{N-i} (i is the level)



Q) Find no. of integers b/w 1 & K. s.t $x \% 10^9+7$ has sum of digits multiple of D.

$1 < K < 10^{10,000}$

$1 \le D \le 100$

Count of $[L,R]$ → following sum property

Pakka digit D.P.!!

So basically $dp(i, t, r, \text{---})$ → sum o/p

- our tree level
- r ensures we r b/w $[L,R]$
- to satisfy our property.

In the above qn we have 10^4 levels. ($K < 10^{10000}$)

$\therefore dp(10^4, 2, 2, 100)$ $D = 100$.

↳ each state has 10 transitions.

$\therefore O(4 \times 10^7)$.

Wat we need to see is that

$dp(i, 0, 0, sum\%D)$

↓

same. if calculated once.

So basically while writing code.

we need to take L, R as strings.

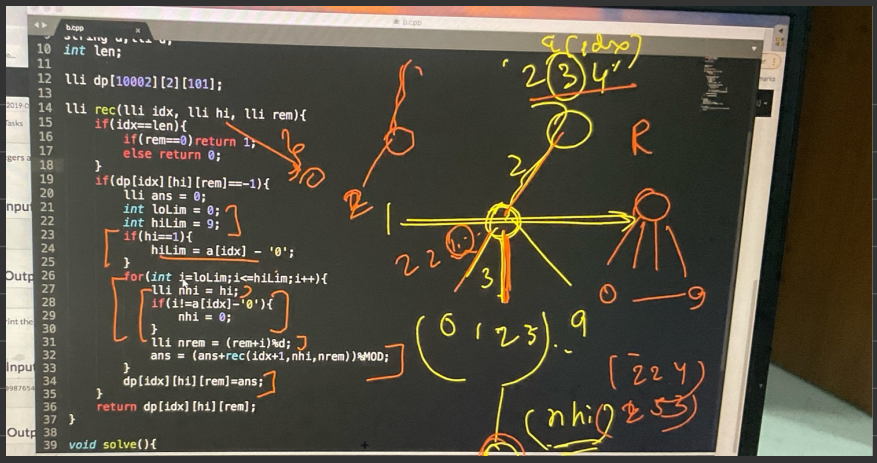
and when hi = 1; we need to take transitions from 0 $\longrightarrow \leq a[idx]$

and we need to update our property variable, new hi/lo accordingly and then call the recursion fn.

actually really beautiful.

So if $[0, K]$ is there we simply dont need to use lo limit. We just use hi in our state to stay towards right of K and substract 1 from our final result.

really beautiful !!



For $[L, R]$ Simply call rec $[1, L-1]$

Same change a, rec $[1, R]$

Subtract both rec$[1,R]$ - rec$[1,L-1]$.

We can't usually do this. So we need to maintain both lo, hi limits.

```
lli rec(int id, lli lo, lli hi, lli rem){
    if(id==len){
        if(rem==0) return 1      → base case.
        return 0;
    }
    if(dp[ ][ ][ ][ ]!=-1){
        loLim=0;
        hiLim=9;
    if(lo==1){
        loLim = aL[id]-'0';
    }
    if(hi==1){
        hiLim = aR[id]-'0';
    }
```

```
for(int i=lolim; i<=hilim; i++){
    nli = li
    nhi = hi
    if(i == L[id]-'0') nli=0;
    if(i == R[id]-'0') nhi=0;
    nrem=(rem+i)%D;
    ans=(ans + rec(id+1, nli, nhi, nrem))%MOD;
}
dp[][][][] = ans;
}
return dp[][][];
}
```

* Property updation step

Dont forget to prepends L with 0's.

Q2) find #of boring nos from [L, R]

↓

digits in even position shld be even & odd position shld be odd.

$1 \le L \le R \le 10^{16}$.

dp[id, li, hi, firstnonzero];

↓

1 — len.

$\therefore$ final ans = $\sum_{i=1}^{len} dp(0, 1, 1, i)$

in lli rec(id, li, ho, firstno){

if (id == len) return 1;

if (dp[x][i] != -1) {

lli ans = 0;

// if (id < firstnonzero num) {

hilim = 9;

}

for(i = lolim; i <= hilim; i++){

if (id ≥ firstnonzero && (id - nonzero - 1)%2 != i%2)

continue;

else recurse.

}

}