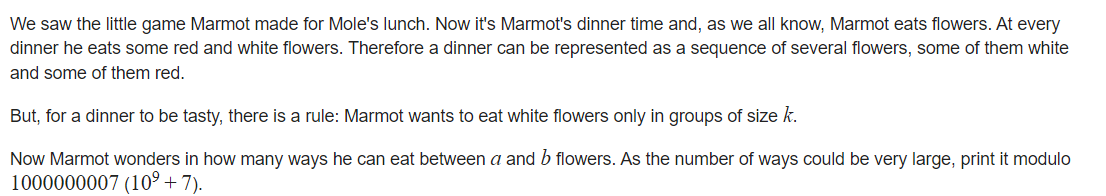
# Changed test

# Send to next state

### Flowers:codeforces 1700\* ( combinatorics) <https://www.youtube.com/watch?v=u13LK6f8wf0&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=3>



Constraint: 1e5

**Soln:**  f(i ) = possible seq of length i( when i flowers are left). If ith point is fixed with red flower, next possible sate is f( i-1).

If ith is fixed by white then, compulsory next k-1 places have to be fixed with white flowers, hence next state is f( i-k) state.( k flowers have been followed by white flowers). **Remain flowers are i- k.**

f(i)= f( i-1) red flower + f( i-k) whote flower

Base case: if i ==k then return 2. 2 ways all white and all red

If i <k then all red, return 1 way.

if( i<=0) return 0.

.

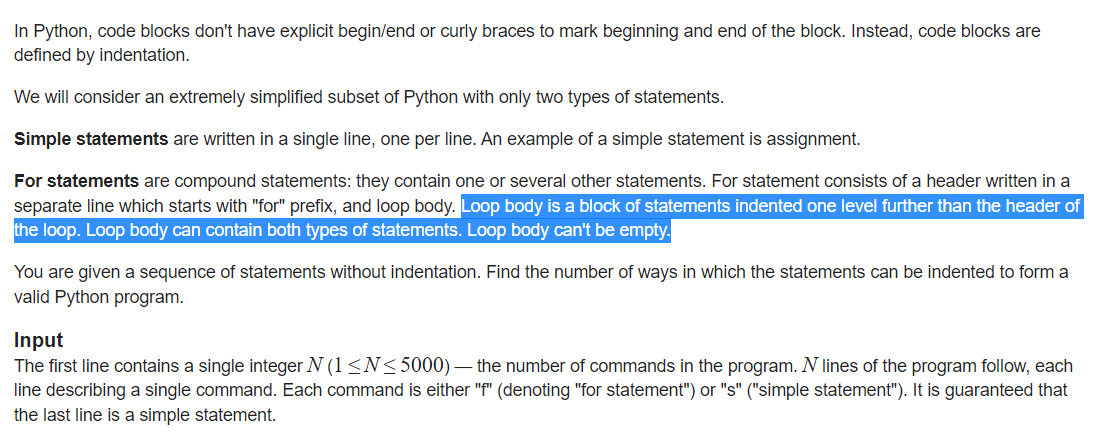
Can store for all i. In O(n). for range queries can use prefix sum.

# 2d dp with optimization

### Python Indentation 455 Div 2 C (Rated 1800)

Soln: <https://www.youtube.com/watch?v=y_hVnH5iR6M&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=6>

Ques:<https://codeforces.com/contest/909/problem/C>



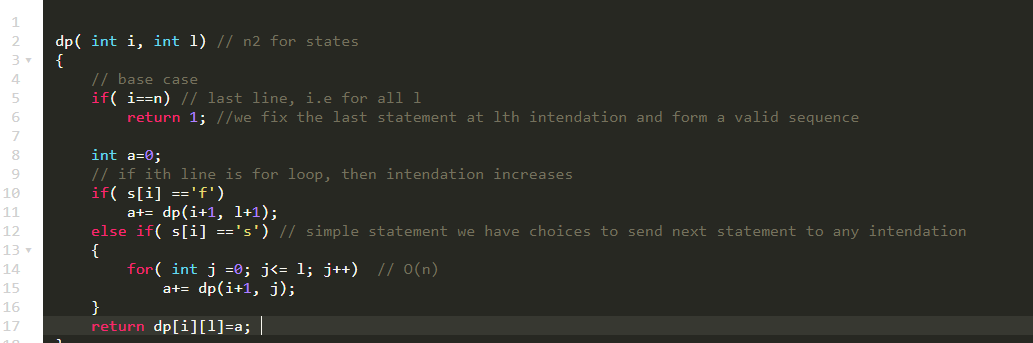
SOLN: to find how many indentations possible. Form a valid indentation and return 1.



States are ith statement and lth level indentation.

dp[i][l]= how many ways are there if we are at ith line and currency at lth level indentation.

Memorization O(N3). Due to loop



Have to do in N2, constraint, we can remove loop, if we make another func.

Or use tabulation

NOTE: when we go in memo from backwards then in tabulation we generally go from i=0 to n

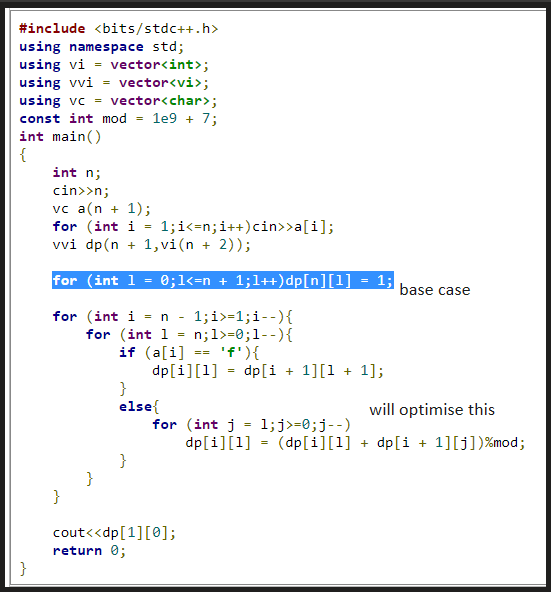
And when in memo we go from left to right , in tabulation we go in in opposite direction.

Here in memo we are going from left to right, that is dp[i] is dependent on dp[i+1]

Hence in tabulation have to go from i=n to 0.

Base case of tabulation is: for all l when i = n ( last statement we return 1

Max levels in indentation =, n lines→ n indentations at max.



Final ans is starting from 1 statement and 0 level.

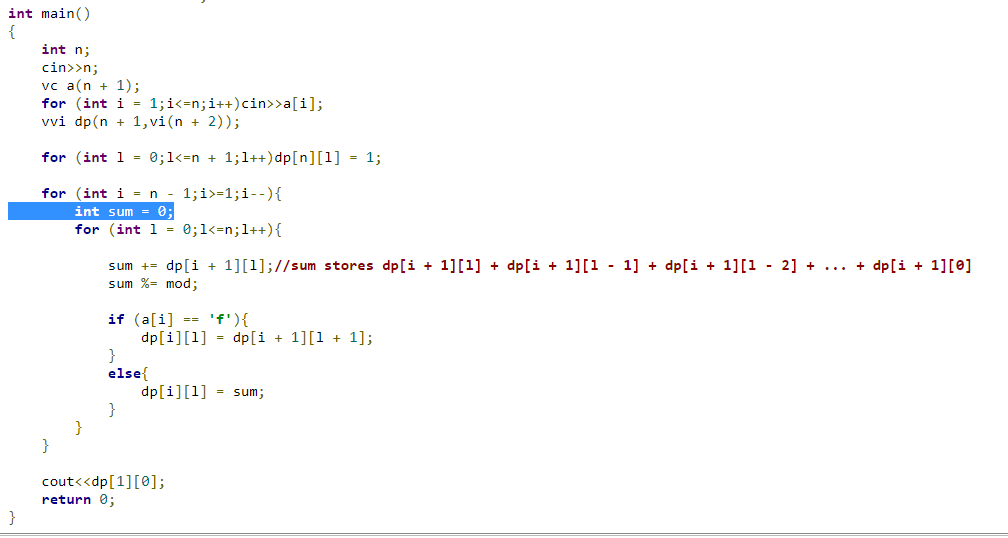
For all nth statement and all levels ans is 1, i,e 1 way.

**For the second loop l can be run from 0 to n, doesnt matter bec we are concerned that to get ans for dp[i] we have dp[i+1]**

To optimise the loop either can do in prefix loop or use the running sum.

For any i, l pair, we wanat sum of dp[i+1] [l from 0 to l].

Hence keeopo running sum.



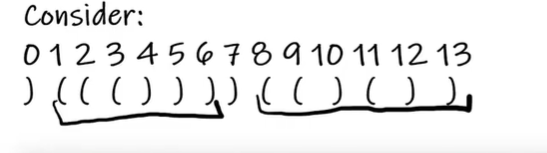
# Brackets

<https://www.youtube.com/watch?v=ola6KSEs3GE&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=8>

**Leetcode /codeforces. codeNcode**

Check for longest regular brackets seq.

()() =4



Soln: use stack, with open brackets going in stack with the index.

For every close bracket pair it with top most open bracket, which forms a valid bracket pair.

Store this length in array, that a clos bracket forms a valid sseq of length l.

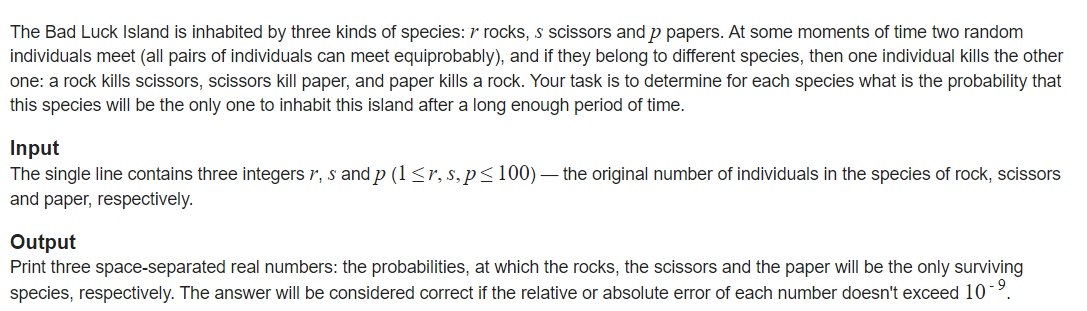
To join the brackets, if current close bracket forms a valid seq of length 4, then go 4 places left. ( o indexed) and check if at this place is closed bracket. And add the lentth of longest seq formed by left bracket.

( () ()).

Find max by iterating over the save array.

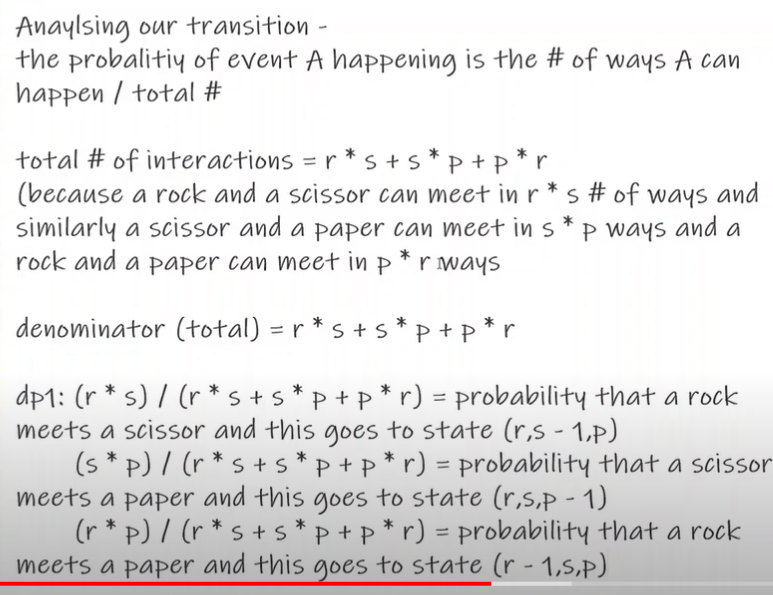
# Probability

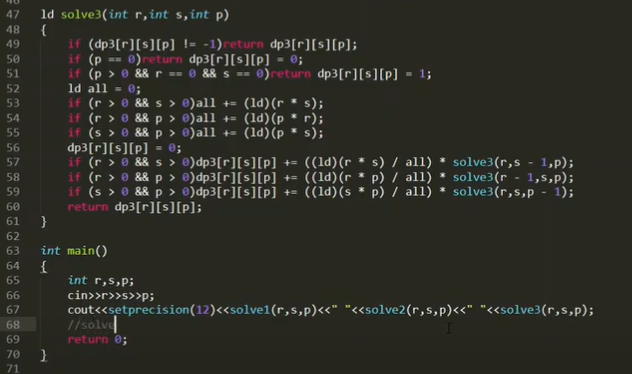
### 3D dp: Ques1: 301 Div 2 D Bad Luck Island (Rated 1900): <https://www.youtube.com/watch?v=wWIMNoVL_z8&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=9>



**Soln:**  to find prop for rock , paper and scissor we can have 3 dp functions.

Dp1[r][p][s] = prob that rock remains at last when we are at this state





# Palindrome

### Count of distinct palindromic substring

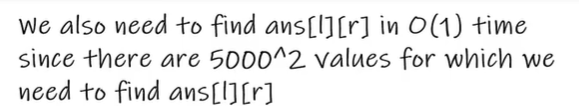
Queries for Number of Palindromes (Rated 1800) **SUBSTRING**

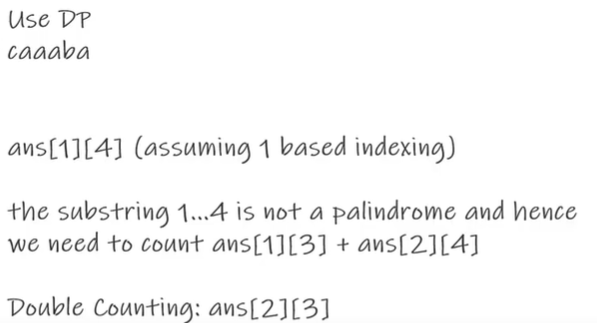
### 

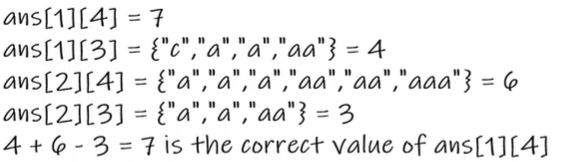
**IMP: Inclusion/ exclusion: Ques:**

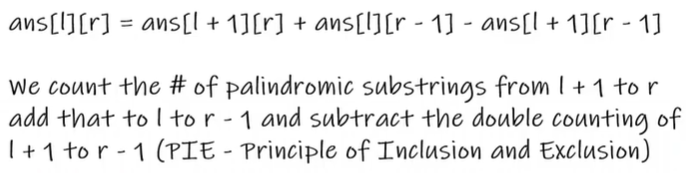
Given string, find no of palindromic substring in each query.. 1 based indexing constrain: string size: 5e3.

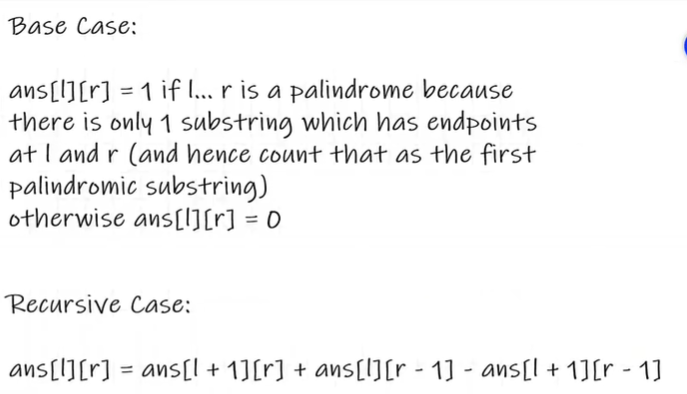
**Soln:**

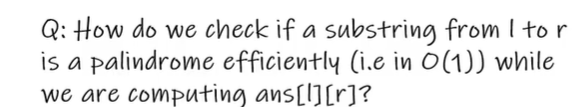


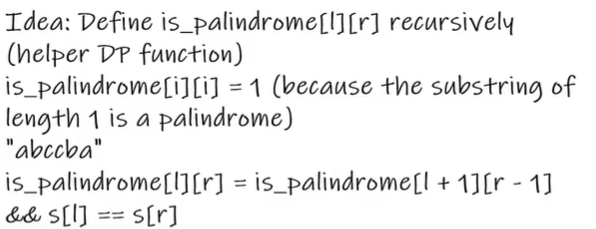


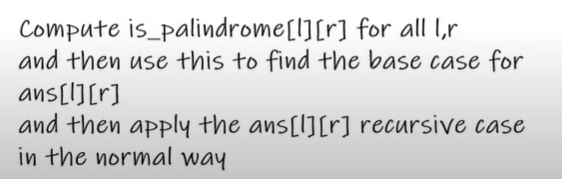


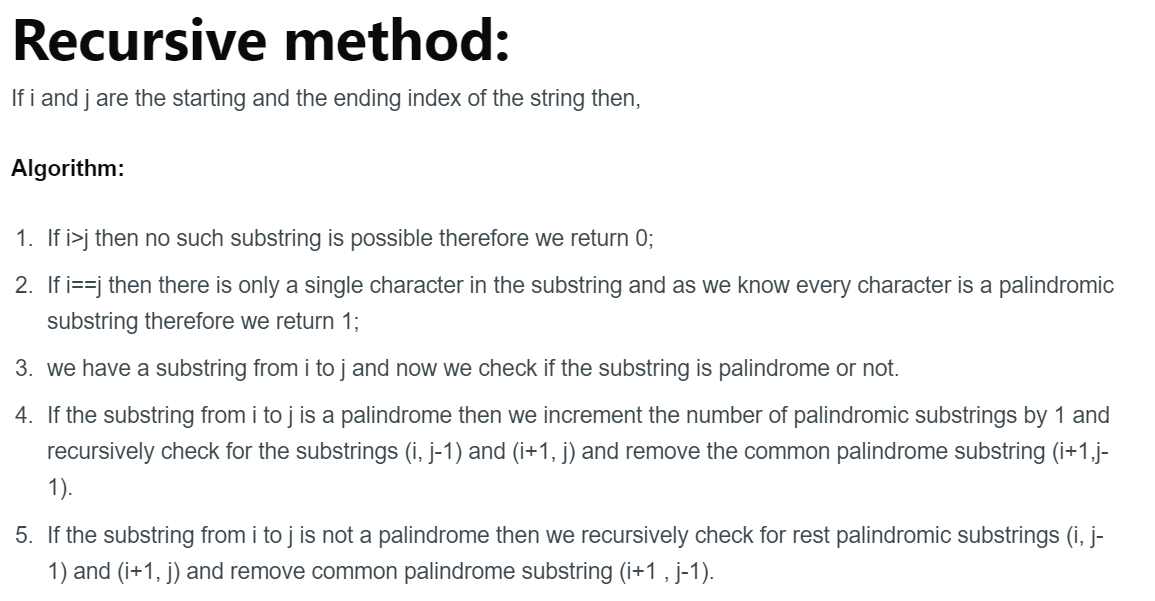




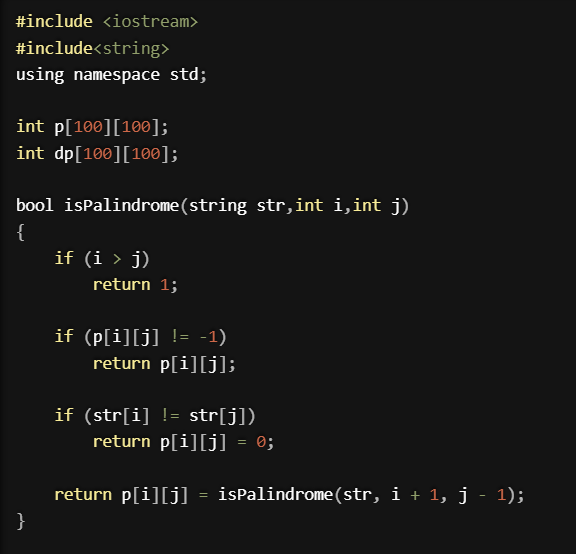








Code: <https://iq.opengenus.org/number-of-palindromic-substrings/>



O(N2)

int rec(int i, int j,string str)

{

if(i>j) return 0 ; // no palindrom possible as invalid substring

if(i==j) return 1; // of 1 length palindrome base case

if(dp[i][j]!=-1) return dp[i][j];

if(rec(str,i,j)) // [l][r]= 1+ lto r-1 + l+1 to r - common substring

return dp[i][j]= 1+ rec(i+1,j,str)+ rec(i,j-1,str) - rec(i+1, j-1,str);

else

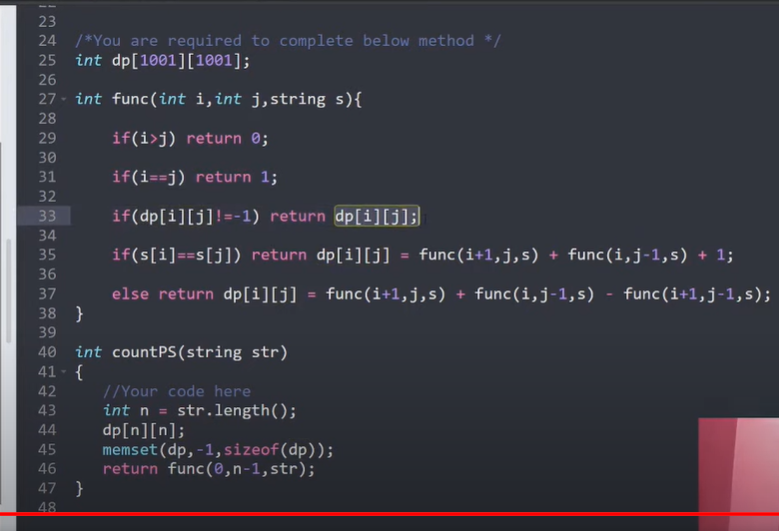
return dp[i][j]= rec(i+1,j,str)+ rec(i,j-1,str)-rec(i+1, j-1,str);

}

### Count of distinct palindromic subsequences

Ques is each query(l,r) have to return the # of distinct subseq in [l,r]

<https://www.youtube.com/watch?v=vlbA8oUxSV0&t=376s>



Abca: when a=a, +1( aa)

Abc + bca ( no subtraction of duplicated bec, for duplicates we can insert a,a at last to from new subseq)

But when i and j aren't equal we cant form new subseq out of duplicates hence have to subtrace.

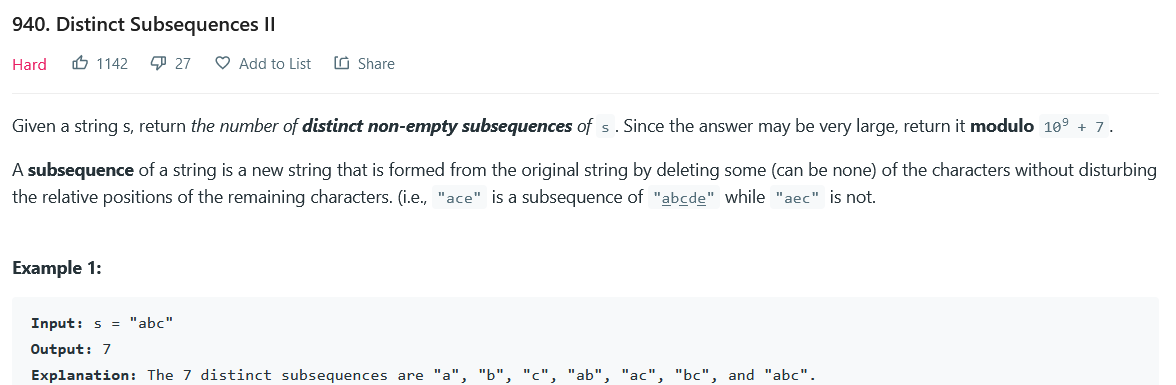
### Count of distinct subsequences(leetcode)

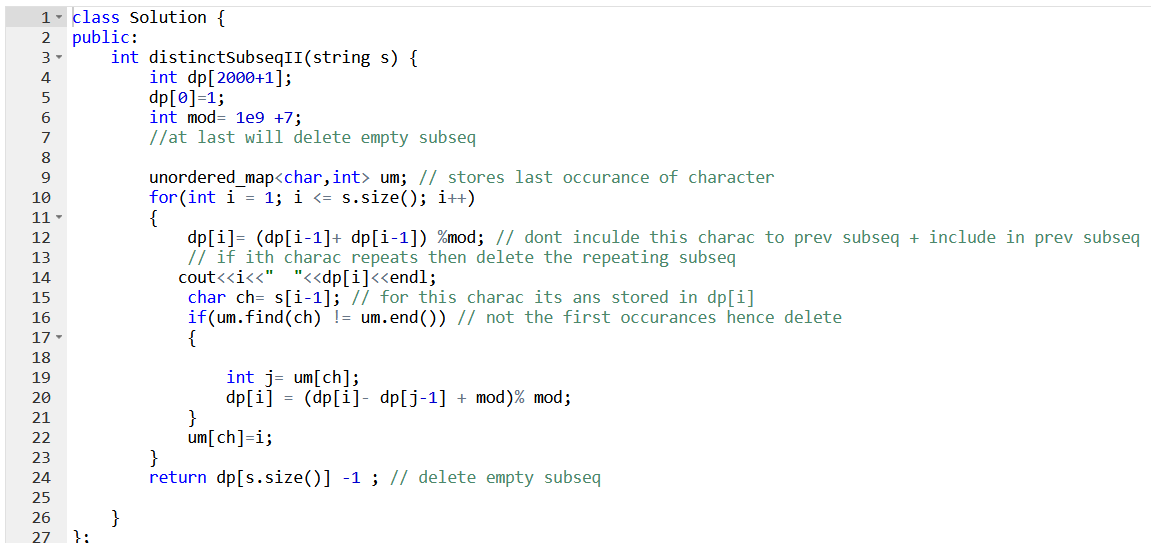
Note: in distinct dubseq we kept dp[l][r]. But here have to return the total count i.e

l=0 always.

Dp[r] = count of distinct subseq from l=0 to r.

There will be again, use of inclusion exclusion.





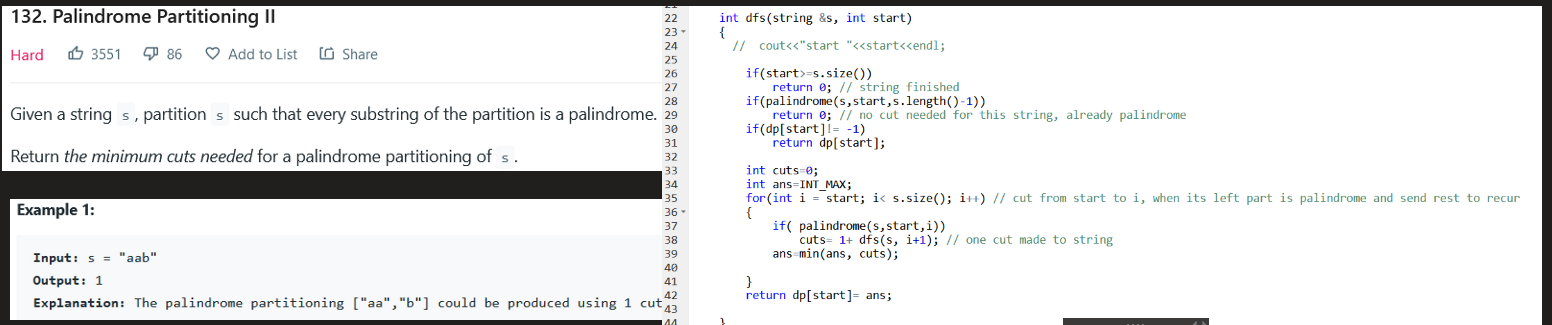
Bec lis fixed at 0. Hence we decrease dp[i] tp dp[i-1] ,if new subseq is possibl;e then add, else delete.

Note: dp[0]=1 is empty string hence

When ith charac is diff rom i-1 charac its same as we did for palindrome subseq, → prev + new +1( here single charac).

### 

### Leetcode

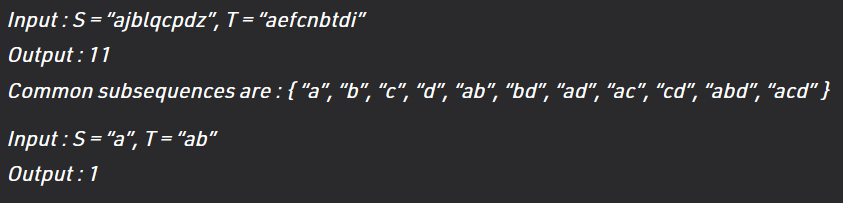


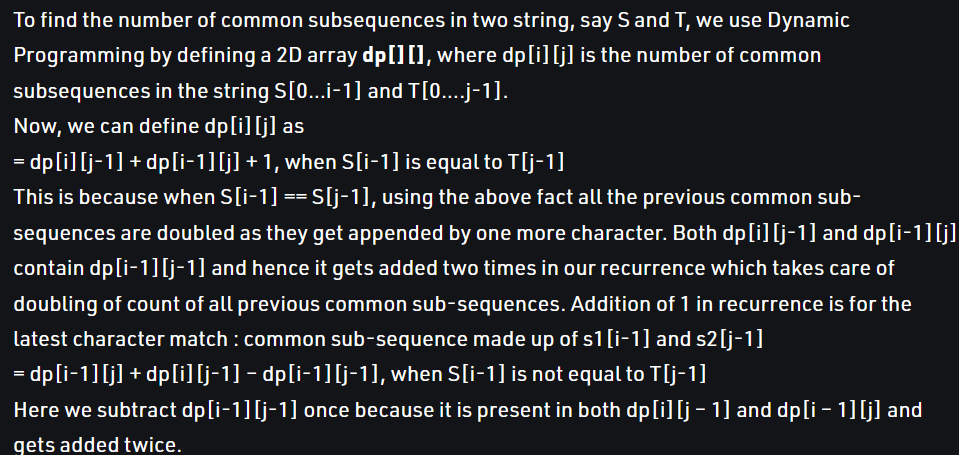
Basic thing to note: add soln from stack when string is finished.

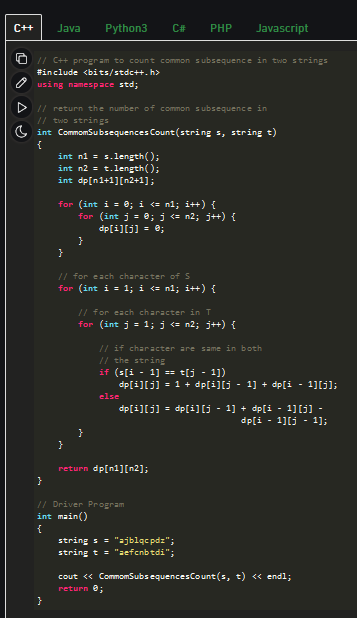
# INclusion/ exclusion

### Count common subsequence in two strings

<https://www.geeksforgeeks.org/count-common-subsequence-in-two-strings/>







# Subsequences /Increasing subsequences

### Longest increasing subseq/ print it

O(n2): all subseq ending at i, store the max one, and index of prev letter.

O(nlogn): at ith index find the max value from 0 to arr[i]. Using bit tree/segment tree/

### Count of k length increasing subseq

O(kn2): dp[k][i]: # inc subseq of length k ending at i.

Base case: for k=1 at all i , =1.

for( l =0to k)

For ( i =1 to n)

For ( j=0 to i)

if( arr[j] <arr[i]

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

Return sum of dp[k][i] fpr all i

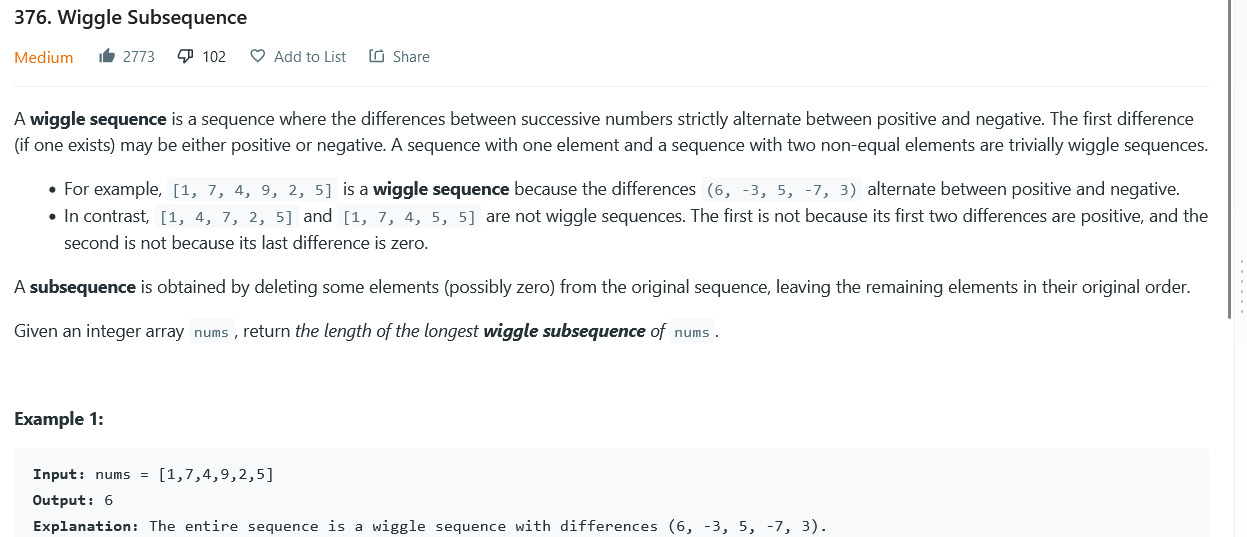
### Count of all increasing sub

Dp[i] : # inc subseq ending at i.

For j =0 t0 i-1, of arr[j]< arr[i] then dp[i]+= dp[j] → O(n2)

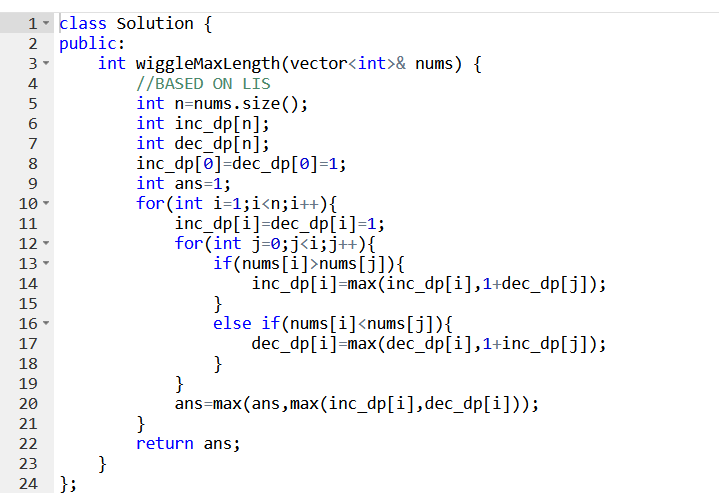
# LIS and variation

### Wiggle subseq leetcode



Soln: self explanatory.

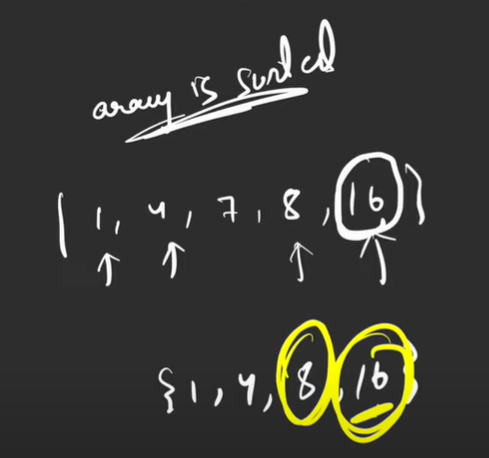
# Low high low high seqs + # high low high low high low sequences



### Largest Divisible Subset | Longest Increasing Subsequence

Largest grp where each pair (a,b) satisfies a%b==0 or b%a==0

Sort the array.



Starting from 4, 1 divides 4

dp[1]=2;

At i=3, for 7

dp[3]=dp[0]+1

and so on LIS

### Longest String Chain

### 

Only 1 letter of insertion or deletion btw 2 consecutive strings

Sort the strings,

dp[i]= let this be the last string.

Check for j from 0 to i-1,

If ( check ( i,j) ) dp[i] = max ( dp[i], dp[j]+1)

Check(st1,st2)

if( st1.size < st2.size()) swap (st2,st1);

n1= st1.size();

n2=st2.size()

if( n1!= s2+1) return false;

while(n1>0 and n2>0)

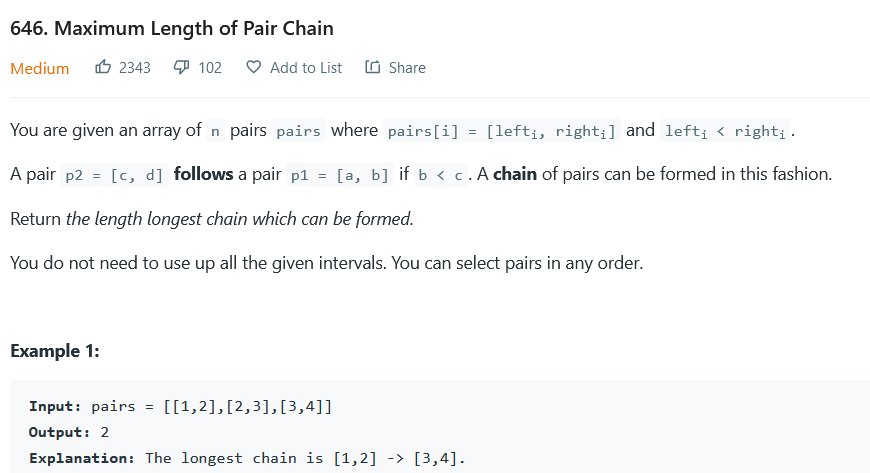
if( st1[n1-1] ==st2[n2-1]) → n1-=1;n2-=1;

else→n1-=1;

if( n1==0 and n2==0) return true else false;

Bec checking if there was only 1 charac diff, then both n1 and n2 should be zero at same time

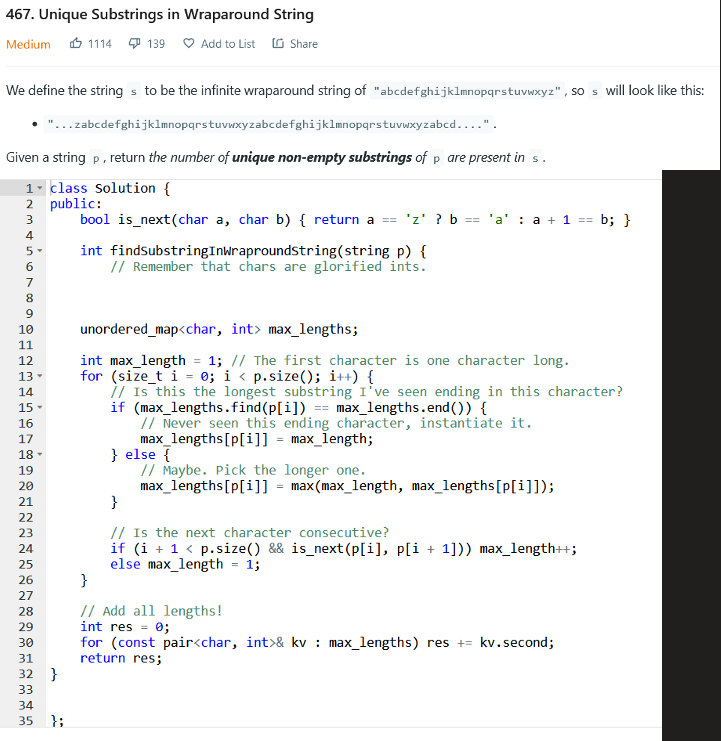
### Leetcode



Soln: self explanatory: <https://leetcode.com/submissions/detail/730360683/>

### Leetcode

Soln: finding longest string ending with every charac will include all substring. And will be unique.



# 

# LCS and variation

### Find length of lcs

Should know recursive and tabulation both

### Print lcs

From tabulation

### Find longest palindrome subseq

Ans = lcs( st, rev(st))

### Min insertion/deletion to make palindrome

Ans = Length - lcs( st, rev(st))

### Convert st1 to st2 , can insert and delete

Ans = convert st1 to st2.

Deletions = st1.size()- lcs( st1,st2) → this will give me only lcs string

Now do insertions

Ans = deletions + st2.size()-lcs

### Shortest string that contain both st1 and st2

Eg brute and groot

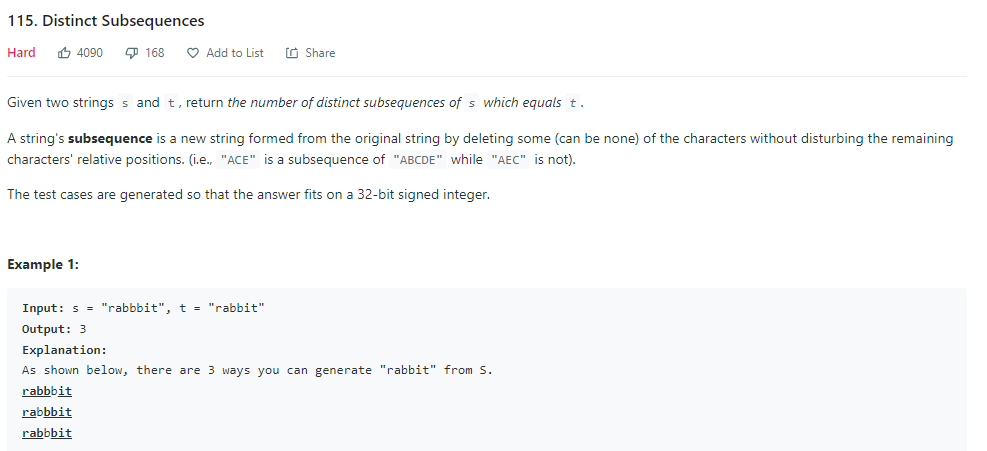
Lcs = rt

Ans can be = b **r** u **t** e (insert the remaining letters from groot)

<https://www.youtube.com/watch?v=xElxAuBcvsU&list=PLgUwDviBIf0qUlt5H_kiKYaNSqJ81PMMY&index=32>

### 

### COunt of subseq of s which are equal to t



Soln: by take u forward.

Start from ending, if match two choices, take or not take.

If do not match, skip this charac.

When size of t becomes 0, found a way return 1.

If size of s becomes, return 0.

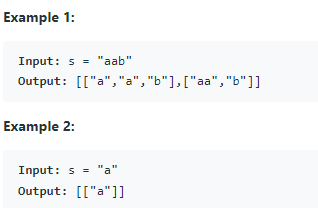
# 

# Partitions possible?

131. Palindrome Partitioning: leetcode

QUES: Given a string s, partition s such that every substring of the partition is a palindrome. Return all possible palindrome partitioning of s.

A palindrome string is a string that reads the same backward as forward.



SOLN:

f( start):

If ( start>=n) // string finished. 1 successful partition

Print or return1 ;

for( i =start; i<st.size();i++)

if( str.substr(start,i) is palidrome)

Push in stack

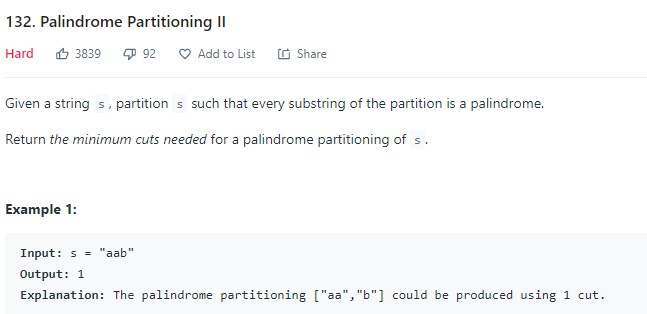
f( i+1)

Pop from stack ( backtracking)

// count or store in stack or whatever

Return ans; // or dp nothing

### Ques palindrome partitioning



SOLN

Like above ques;

Base case if string is palindrom

Return 0; no partition needed

Int p=INT\_max

for( i , start , n)

If ( st[i to n) is palindrome)

p= min( 1+ f( i+1)

// atleast one partition is always possible a |b| c| d →palidromes

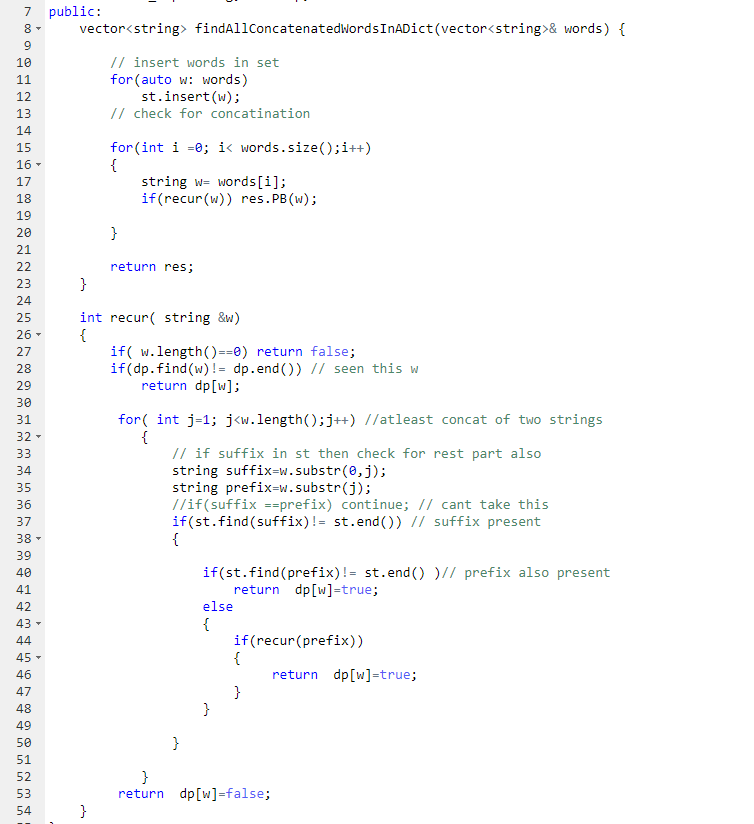
### 

### ques



Soln:

<https://leetcode.com/submissions/detail/672444458/>



Note: have to check for atleast 1 joint,

Hence check if prefix and suffix both exists return true then and there only

s= w1+w2+w3

Break at w1→ r( w2+w3)( returns true)

# Count of uniques substring/ subseq?

For count o increasingf subseq? Its same as count of increasing subseq

### 

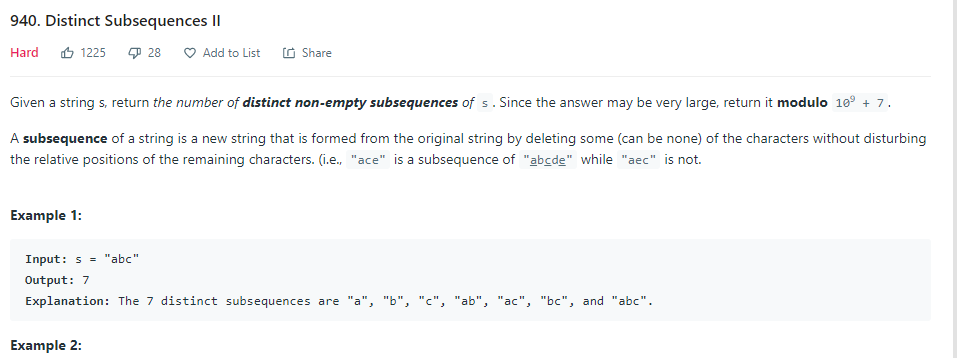
Soln: find longest substring that end with a/b/c/d/… all 26 characters.

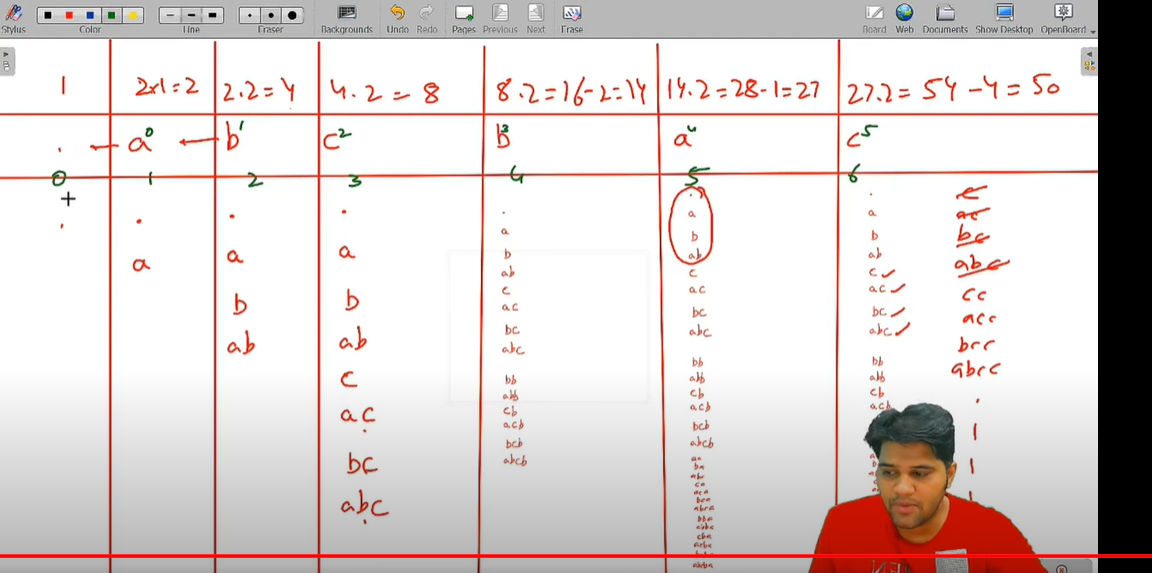
Ans = sum of dp[ a]+dp[b]=...dp[z]

Let the longest substring ending with c is

Xyzabc → diff substring possible are xyzabc , yzabc, zabc, abc , bc , c

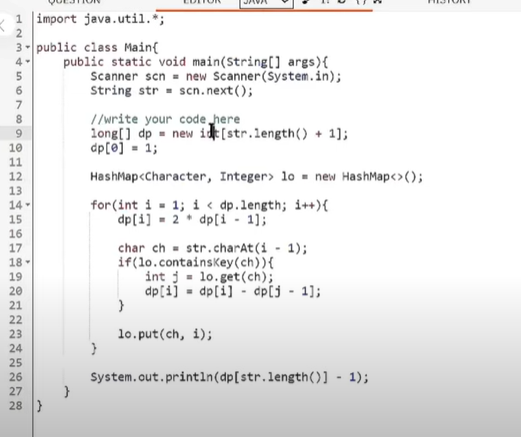
### Count of unique subseq





Famous ques soln by pepcoding

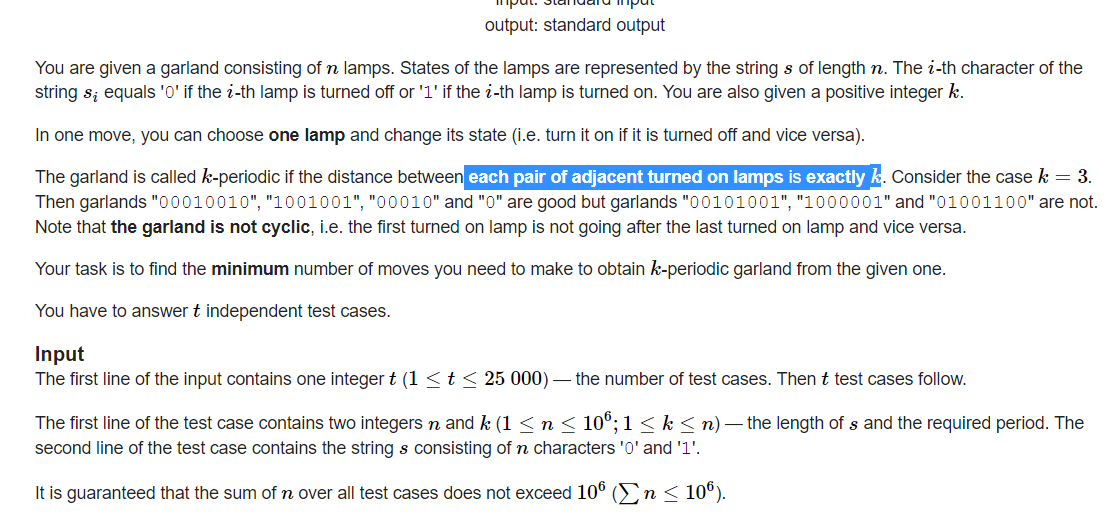
<https://www.youtube.com/watch?v=9UEHPiK53BA&t=990s>



# IMP:Fixing one place imp

Ques: <https://codeforces.com/contest/1353/problem/E>

K-periodic garlands (rated 1900)



Soln: due to 1e6 constraint, either n or nlogn

One possible soln is close all. No ones. This is one of the ans.

Only one lamp is on

Condition is every pair of adjacent should be k dist apart.

Its similar to form the longest AP seq. Where we fix the first elm of seq.

Here we fix the first or last occurrence of 1.

If we fix i as last occurrence of 1. Then everything right of i is 0( add required switches to make everything 0) on kth left step . we have choice to keep it on, or if we keep that off then everything remains off left onwards.

2 choices

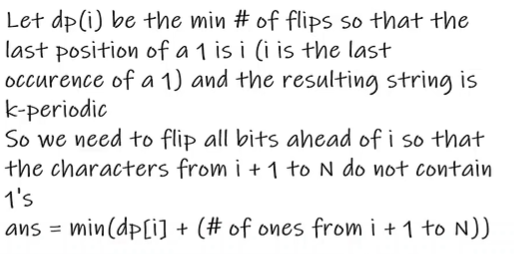
Dp[i] = min( only ith lamp on, everything to its left is off **OR** we switch on the prev i-k lamp on and let it decide best for it dp[i-k] + to witch off all lamps btw the immediate prev and ith lamp)

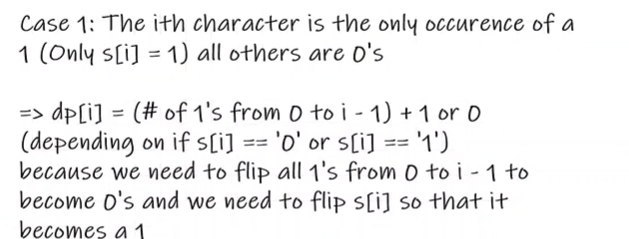
Dp is created only for left side , right side is added at last

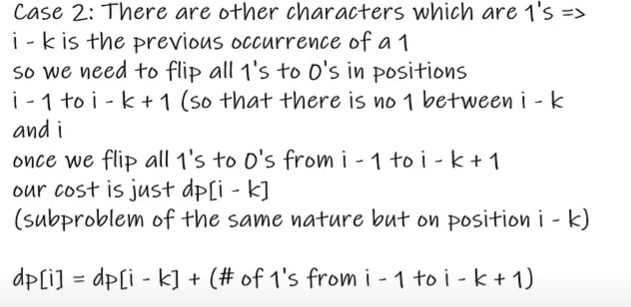
**Or edge case**

if want to keep only ith lamp on then every other 1 needs to be 0. +1( if ith lamp is off).

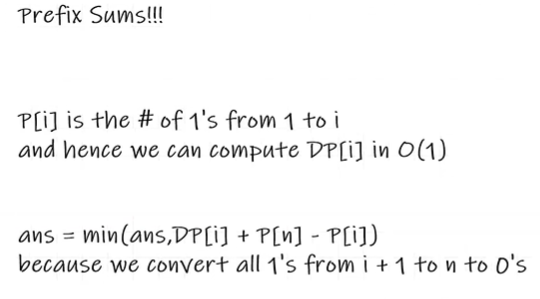
Calculate required moves and add.



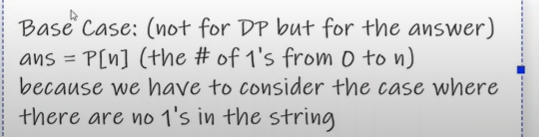


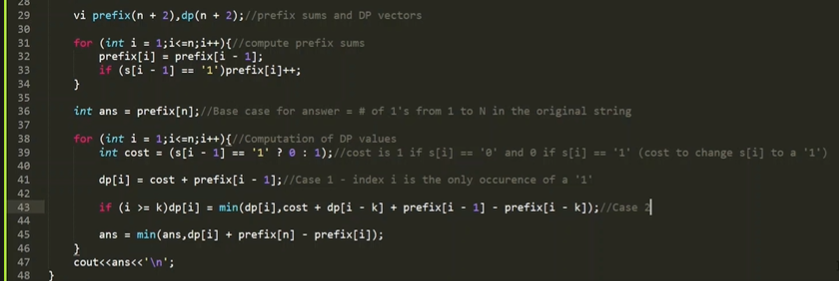
****

**Optimization**

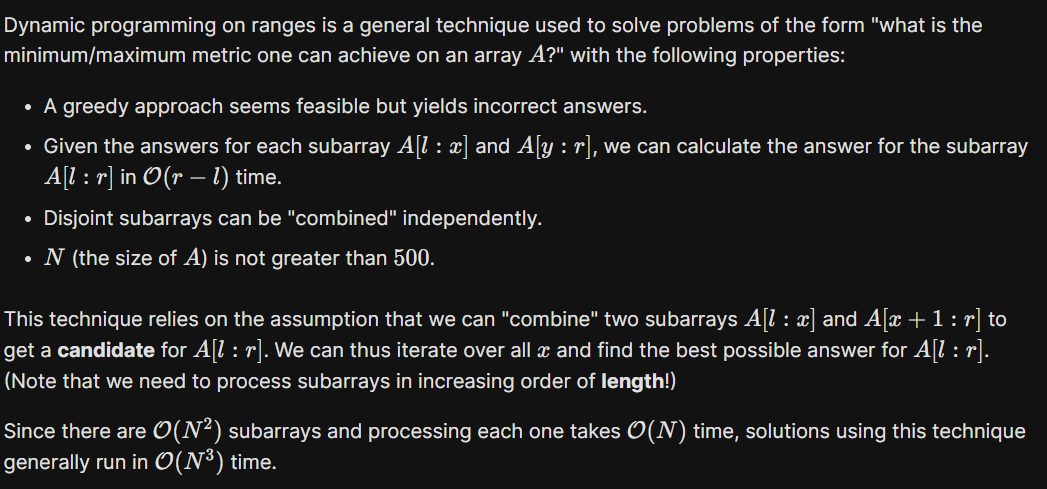
**\**

**Edge case: no lamps are on**

****

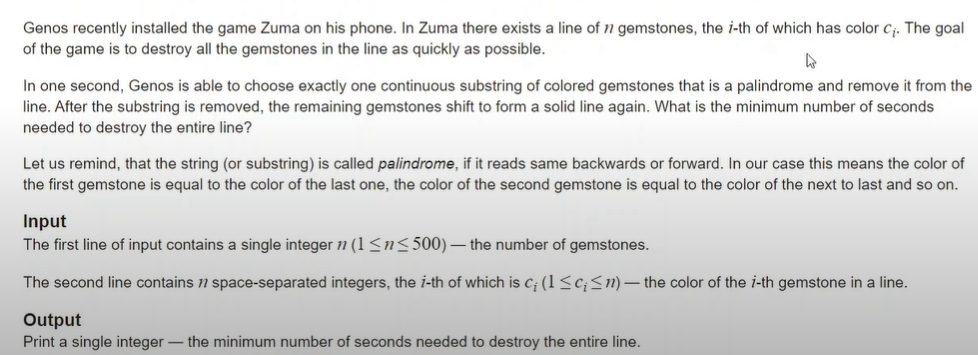
****

# Range DP



### ques 1 ZUMA rated 1900

<https://www.youtube.com/watch?v=euix55irit4&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=12>: adhish solution



**SOLN: ans is dp[0][n-1]**

int getAns(int l, int r)

{

if(l > r) return 0; // invalid seq

if(l == r) return 1; // single charac, is palindrome can be deleted in 1 move

if(dp[l][r] != -1) return dp[l][r]; // caching

int tem = inf;

tem = min(tem, 1 + getAns(l + 1, r)); // delta the leftmost alone

if(a[l] == a[l + 1]) tem = min(tem, getAns(l + 2, r) + 1); // delta leftmost with adjacent one as palindrome of length 2

for(int i = l + 2; i <= r; i++) // if leftmost is decided to be deleted as part of a palindrome

{

if(a[i] == a[l]) tem = min(tem, getAns(l + 1, i - 1) + getAns(i + 1, r)); // l and k can be deleted when delting the last charac of middle string

}

return dp[l][r] = tem;

}

int solve()

{

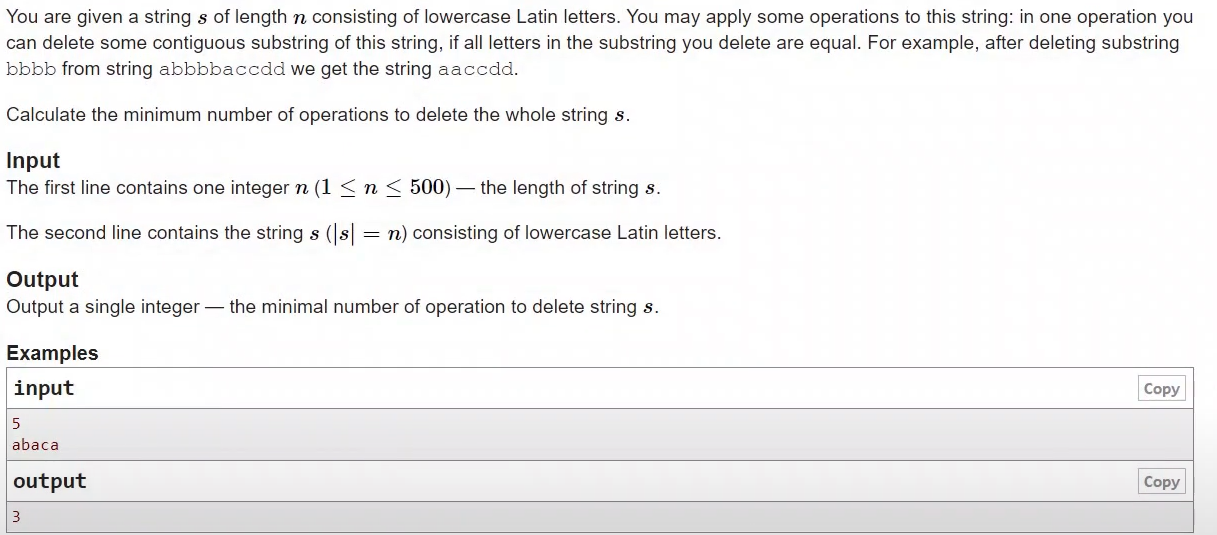
memset(dp, -1, sizeof dp);

int ans = getAns(0, n - 1);

return ans;

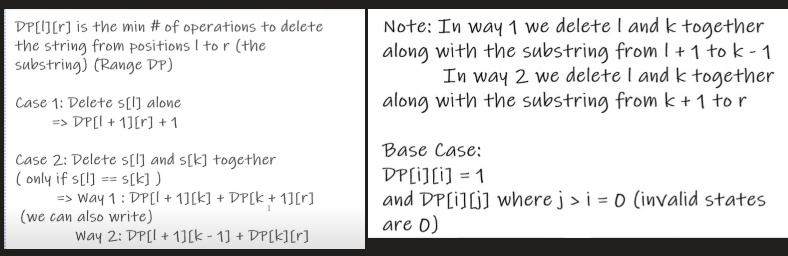
}

### Ques : 61 Div 2 F - Clear the String (Rated 2000)



**SOLN:**





Note: for

aa l =k then l+1>k-1, and returned 0.

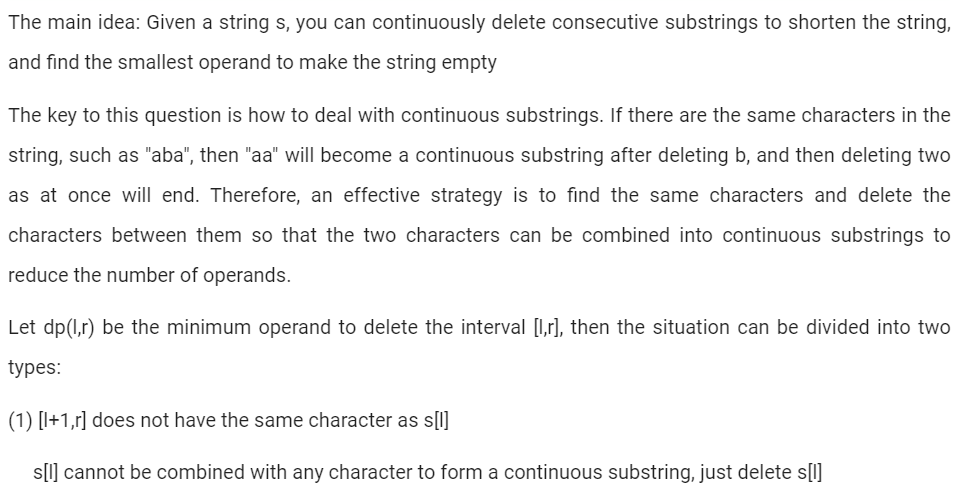
But we also have dp[k][r] ( way2) that is deleting l with k. Which returns 1 and give correct ans.

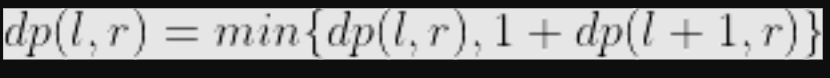
No need for deleting l with l+1 when both are same bec. While calculating for k=l+1.

We delete l and k together(dp gives 0 , goes to invalid state) + dp[k][r](deleting leftmost with k which os here immediate neighbour)

Even if we keep this makes no diff( bec founding minimum)

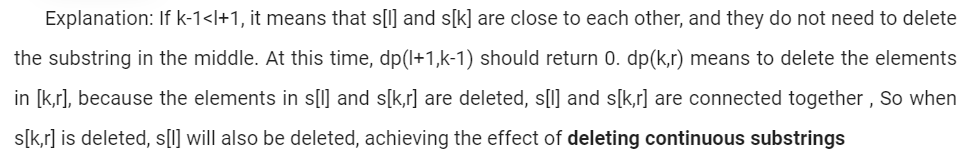
<https://blog.karthisoftek.com/a?ID=00950-0585b92b-f45d-47f6-bcd0-e8e94111d346>.: blog for further explaination











Ques 2 codeforces road optimization

Ques 3 remove boxes

Ques 4 strange printers

# Grid

### Cherry blossom1 leetcode

### Cherry blossom2 leetcode

### 1600 to do <https://www.youtube.com/watch?v=-oFiDyvkIOs&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=17>

imp:The Least Round Way (rated 2000)

<https://www.youtube.com/watch?v=M3jSF8y6otI&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=15>

### 

Soln:

Need least no of zeros.

Edge case: if there is 0 in a cell then. One way is to have one zero in our path.

10 can be made with 2\*5

Hence in our path the no of zeros are min(2,5).

We cant start from a X cell and reach the bottom and check min zeros from that cell bec, ans will depend on the path taken before reaching the X cell.

but we can divide the ans as. Path having 2’ abd path having min 5.

In this starting from any X, contribute to same level of no of 2’s bec simple addition.

.>

Approach: find min 2’s path in g1, →k1

Find min 5 path, in g2. →k2.

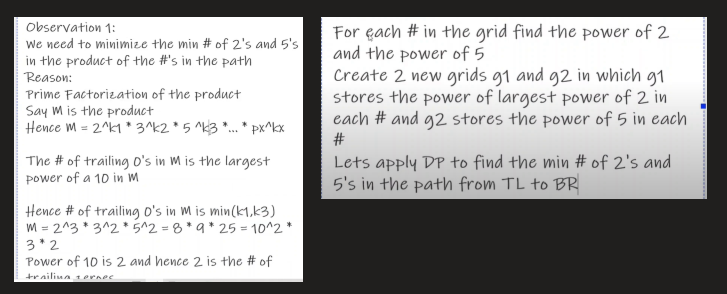
Return min(k1,k2)

Why? We dont need to b-find the no of 5 in path of 2? And vice versa?

Bec say k1 =20 that means , 20 twos were there in min path. And if fives in this path were also minimum , then g2 path will be this only.

And same for k2.

Hence finding min path, in both cases independently will take care of every case.



How to find power of 2 in prime factorization of a no (say E)

while(e != 0 and e%2 ==0) // divisible by 2

{ cnt++;

e=e/2; // wil give int only bec already checked if e was div by 2. Then only came in while loop

}

Similarly for 5.

Do this for every no. log(2).

For min 2’s path having zero. Let it be. If the no of zeros comes >1 and 5’s also.

Say min ( 2,5)=1 that means atleat 1 zero in the product is there. Then take a path having 0 in grid cell.

And if min(2,5) =0 but in the path there was a cell whose value =0, can maintain a flag for every cell.

If the path having zero is inevitable. That is say there is a path which have 0 two’s then just check if that path has 0 in a cell or not.

And for that no need to store the path just maintain a flag variable for every cell.

The min #2 from right or left which is selected its flag value is copied.

# Kadane’s

Ques: a subarray is chosen

Step2: largest element from this subarray is removed.

And the sum of that subarray is calculated avoiding the removed no. find the largest possible sum of subarray after 2 steps.

Constraint : 1e5 nos, and each no is from -30 to 30

**SOLN:**

Ap1: all subarray ending at i, find the max size subarray, and the max no, in that subarray. ( kadane’s) if the size of this subarray is 1, then removing the positive element will will lower the subarray to 0 if only element is +ve. Else max element isnt removed.

If all negative then score is 0.

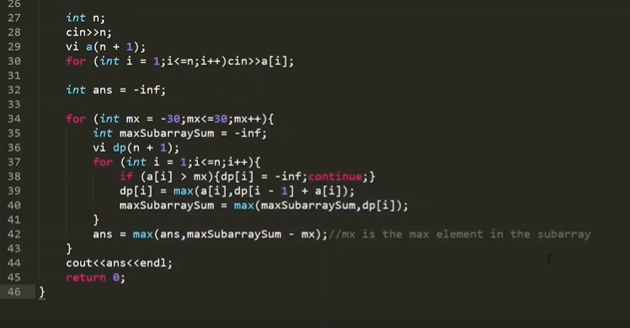
AP2: bec the elements are in -30 to 30 hence we can use it in logic:

Let mx be the mac no in subraay that will be removed.

Note: if a subarray doesnt have mx still removing it wont effect the final ans. Bec for the final ans, if from subarray 10 is removed, then going through all possible cases of mx, we will get subarray.

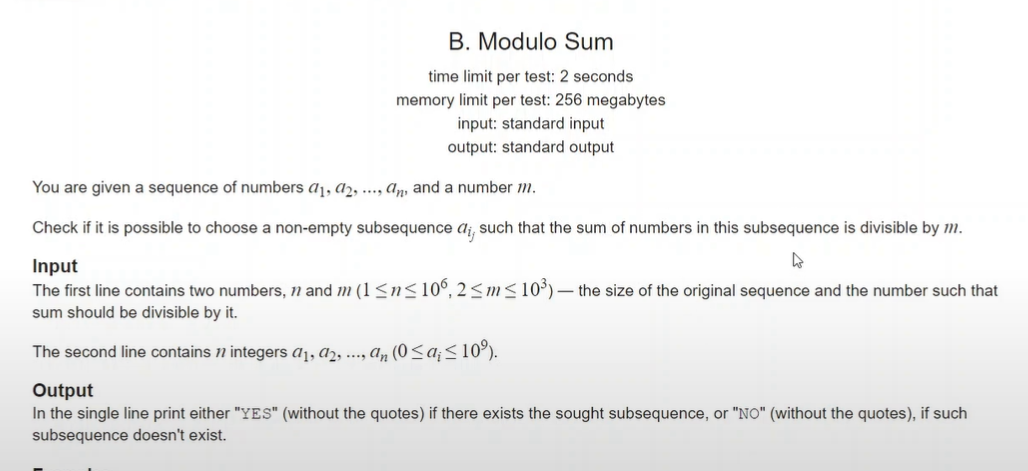
**For negative no. max subarray is say -3, we remove it -3-(mx=-3) we get 0.**

<https://www.youtube.com/watch?v=KMOX3XIovKM&list=PLzVLIdIx9dQxwAN5mMkzdvK2B4rV8879j&index=16>



# Pigeon hole: classical problem

GFG, coeforces 1700



Soln:

For div means modulo m =0.

If no of elements are >=m output is yes.

Prefix sum ending at i can be 0 to m-1( possible remainders) to make it div by 0, we only need

prefix( i1) - prefix(i2)=0.

prefix(i1)%m ==prefix(i2)%m ( and this will happen for sure acc to pigeon theorem)

**For n<m( knapsack prob)**

m=1000 then n is also 1000( only n<1000)

1e3, can work in O(n2).

We need a subseq having sum with mod m.

Convert all nos to arr[i] %=m

Have to find if we can find sum of subseq as m

//base case

If sum =0

Return 1;

If n<=0 return 0; // false . nos finished

// no need for negative sum, here, though its there in knapsack

If we take no

Bool a=dp(n-1, (sum-arr[i]+m)%m); // bec taking mod in sum hence sum be m/2m/3m doesnt matter.

If we dont take it

Bool b= dp(n-1, sum)

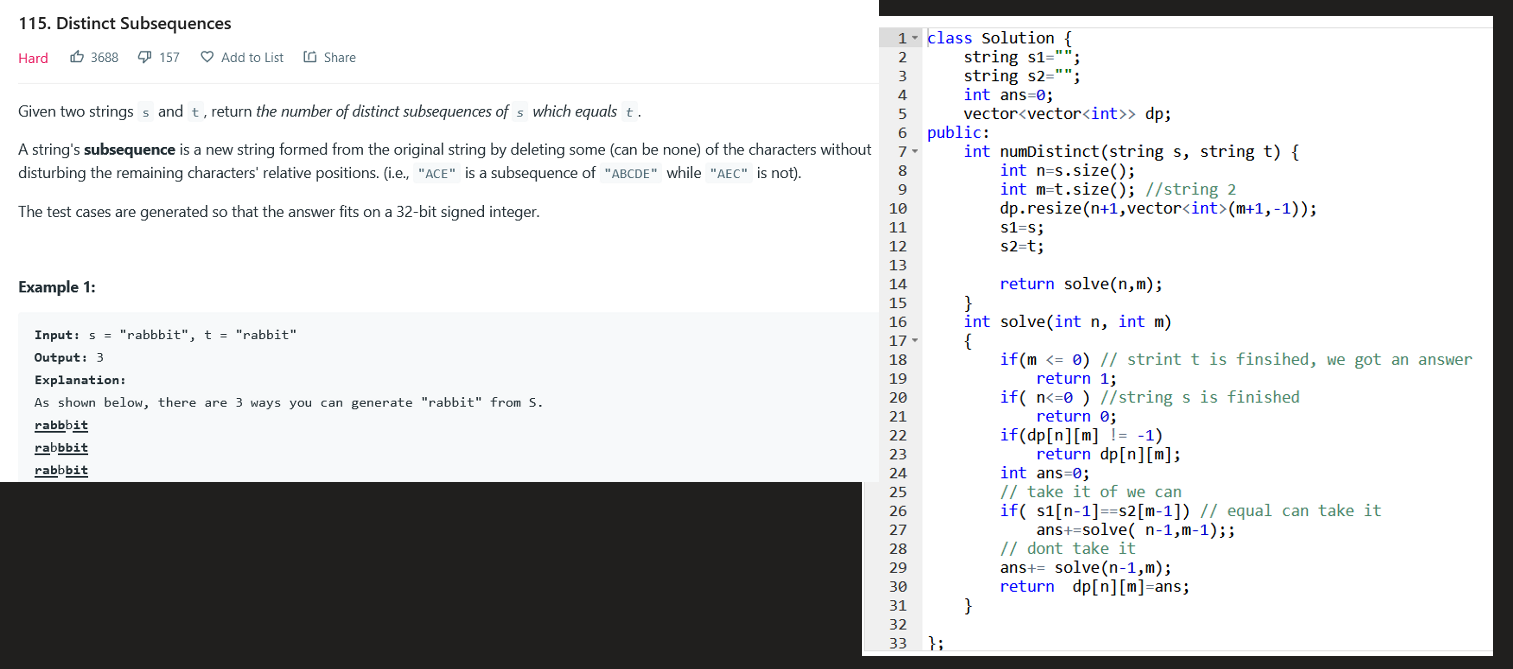
Return a or b;

# To choose or not to choose

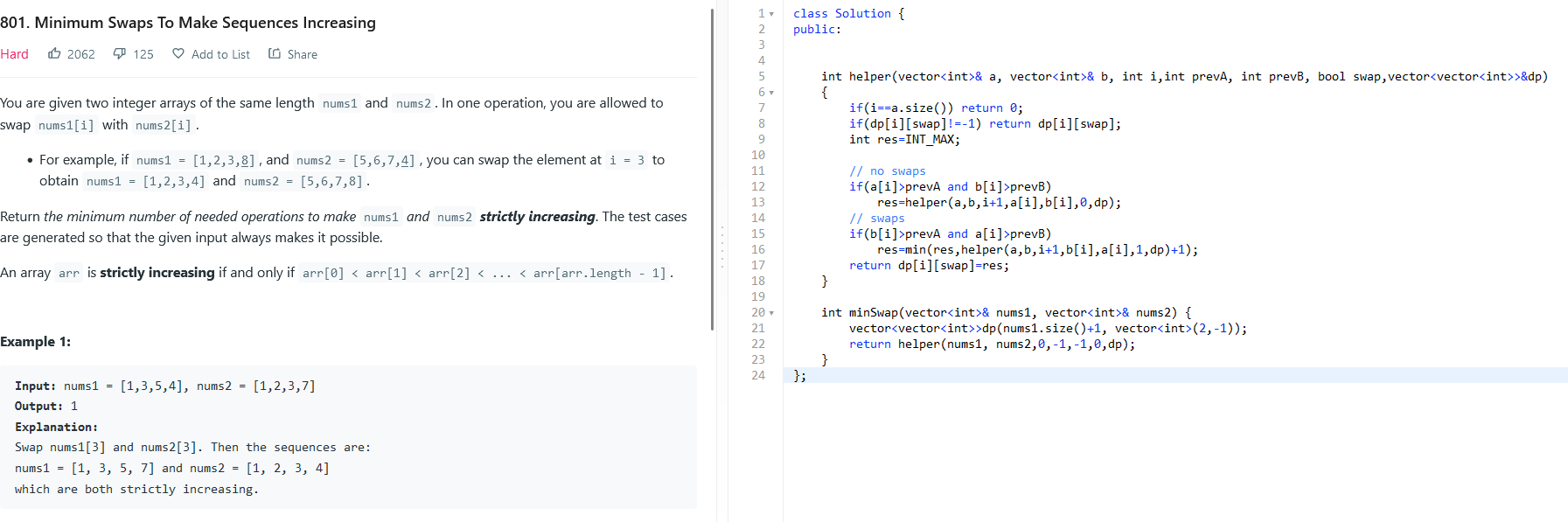
Video 19 onwards

# Take it or not take it

Leetcode ques



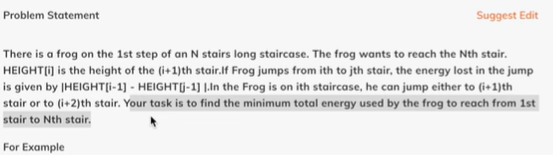
### Leetcode



SOln: swap it or dont swap it.

## Frog jump/ ways to reach last stair/no

### Ques 1( min way) basic:



Soln:

Dp[i] = min( dp[i+1], dp[i+2])

Recursive:

f( i ):

if ( i ==n ) return 0;

If in cache return cache;

Return dp[i]= min( dp[i+1]+ cost1, dp[i+2]+cost2);

LOOPING

dp[i]= the min possible from ith stair

dp[i] uses future value, hence looping in reverse manner.

For ( i = n-2; i>=0 ; i–)

dp[i]=min( …)

**Only using last 2 values hence just use variable, array not required.**

### Part2: if the jumps are possible from 1 to k;

For ( i = n-2; i>=0 ; i–)

for( j,1,k)

if( i+k<=n ) dp[i] = min( i+k)+cost;

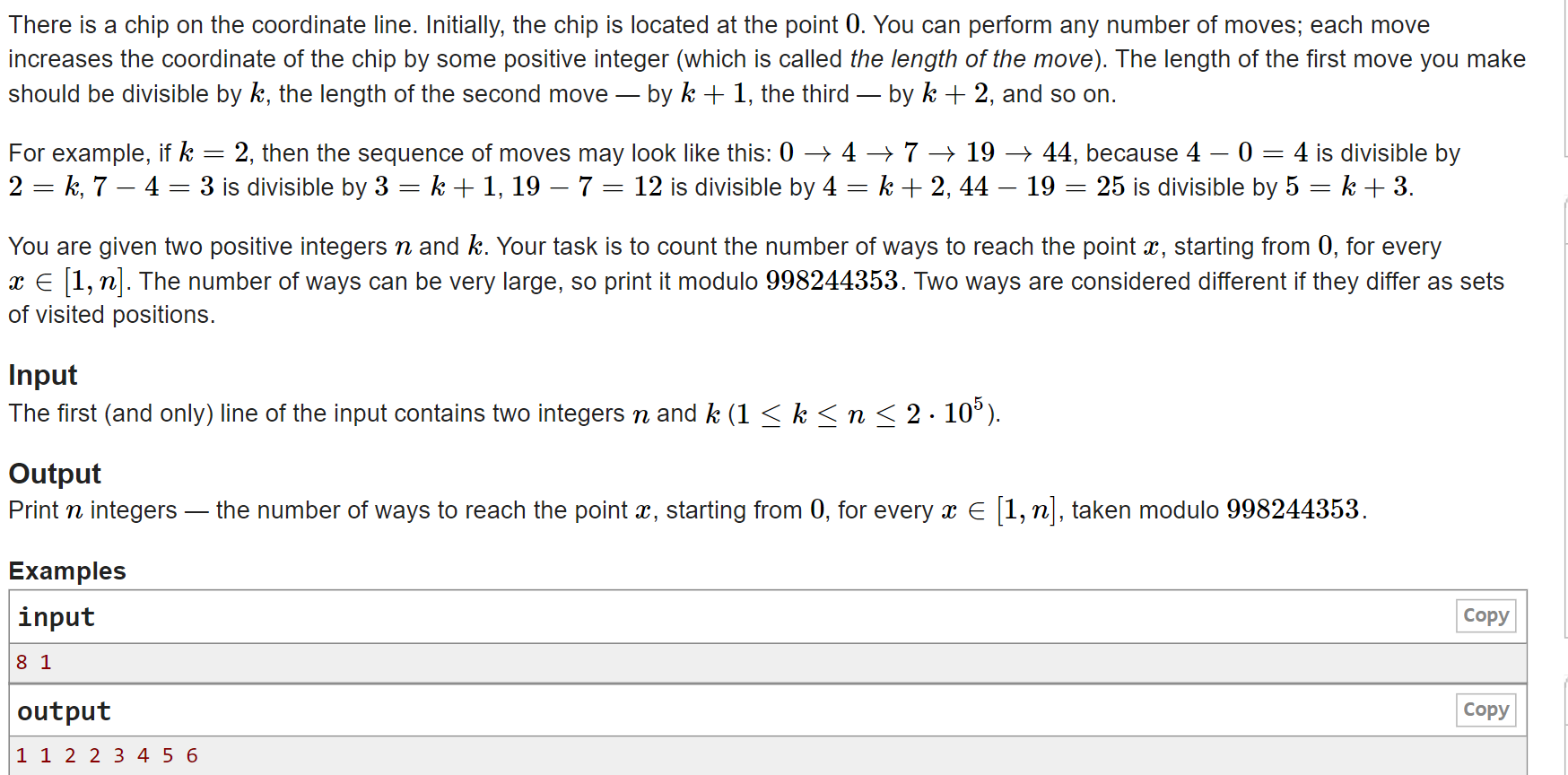
Optimization possible bec we are calculating min

### Ques 2 find ways to reach last stair: iMP

Codeforced D 133 educational round **CHIP MOVE**

Ques:<https://codeforces.com/contest/1716/problem/D>

Soln: <https://youtu.be/ppqurybgZrQ?t=1729>



Soln:

There is hint in the ques given, we have to give ans for every x from ( 1,n).

Max moves possible are ( 1+2+3..m) =n

m=sqrt(n), n=1e5

m<=400 ( upper bound)

Recursive soln ( NAIVE SOLN):

If we only need to find ways possible to reach lasst nth spot.

dp[m][i]= moves possible from ith spot to reach nth spot such that already made m moves.

Dp[m][i] = dp[m+1][ i+m] + dp[m+1][ i+2m]+dp[m+1][ i+3m]+dp[m+1][ i+4m]+... till i+xm <=n

f( m,i):

For( j= i+m+1; j<=n ; j+=m+1) // takes n/3 for third move and so on

Ans += dp[m+1][j];

Return dp[m][i]=ans

O( sqrt(n) \* n\* something)--> TLE

SMART APPROACH

Taking hint from ques also thinking in opposite way.

Have to find ans for every x, not only ways to reach n from sytating pont.

Dp[m][i] = ways to reach ith spot in m moves

if we reach i in m move that is we made mth jump in multiples o k+m-1

x=k+m-1

Dp[m][i] = dp[m-1][i- x] + dp[m-1][i- 2x] + dp[m-1][i- 3x] + dp[m-1][i- 4x] + .. ( this is pull DP) ( bec of this we have a summation computation which we can optimize)

For push dp

At every dp[m][i ] update all dp[m+1][ i+x] = dp[m][i ]+1; ( one way ) // like in minimum taps to water the garden

// wont be able to optimize this but.

First thing have to optimise space by storing only i and i-1 row. Else MLE. **(** **Memory limit exceeded).**

Prev[i]; // all zero

Ans[i];

ans[0]=1;

for( m =1; m<= 400; m++) // upper bound

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

for( k =1; k<=400; k++)

if( i - (k+m-1) <0 ) break; // if not possible to go back break

Dp[i] = prev[ i - (k+m-1) ];

ans[i]= + = dp[i]; // adding ans for the mth move.

// before starting new move

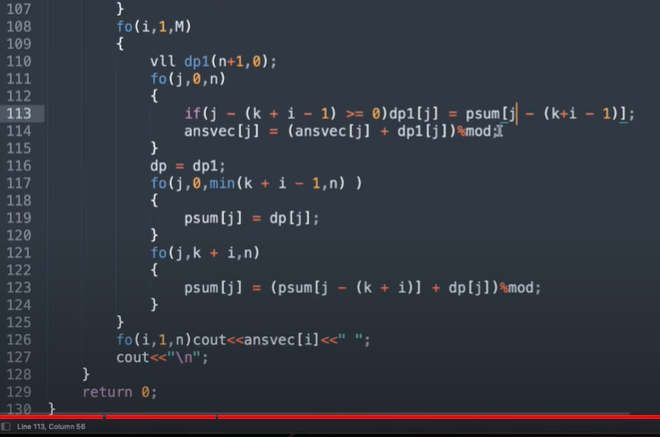
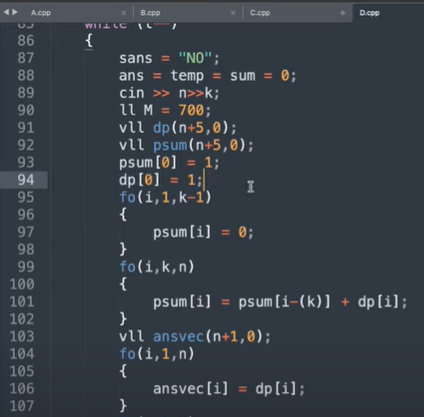
Prev =dp;

Dp.clear()

print ans[i]; // this will give count of ways to reach ith spot after completing m moves.

TLE: same as above 400\*n\*something+ 400\*2\*n

***OPTMIZATION: maintain prefix array top avoid third nested loop, and for computation dont use nested loop***



In the 1st move we can jump from 0 to k/2k/3k/4k ( multiples of k)

*Hence in line 100 maintaining prefix sum for 0th move.*

Where prefix[0]=1; initialised

For the prefix just before every move precompute the prefix

Prev[i]; // all zero

Prefix[i];

Ans[i];

ans[0]=prefix[0]=1;

for( m =1; m<= 400; m++) // upper bound

vector<int> dp(n+5,0);

for ( i =1; i<=n; i++) // for ist spot, as 0th spot is always 1

if( i - (k+m-1) >=0 ) dp[i]=prefix[i - (k+m-1)];

ans[i]= + = dp[i]; // adding ans for the mth move.

// before starting new move

// copy everything)

Prefix = dp;

// maintain prefix sum for this move it will be used in next move when m=current m+1

For ( j =k+m; j<= n ; j++)

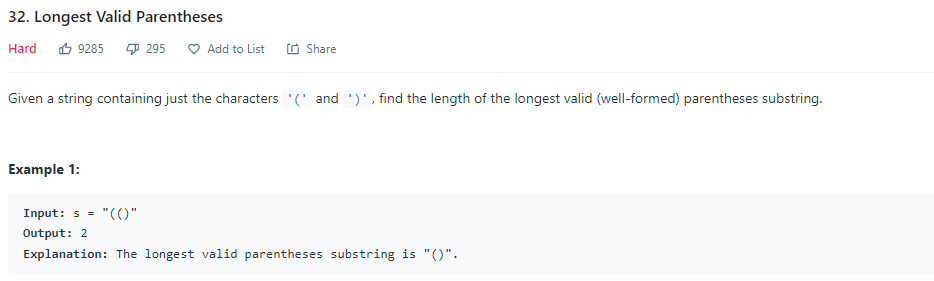
Prefix[j] = prefix[ j - (k+m) ] + dp[j]; // ( previous + current) ( prefix[i] = upto ith term

// print ans[i]

Complexity: sqrt(m) (n + n + n)--> 400\*1e5 \* 4= 1600\* 1e5→ 1.6 \* 1e8= **1.6 secs** ACCEPTED

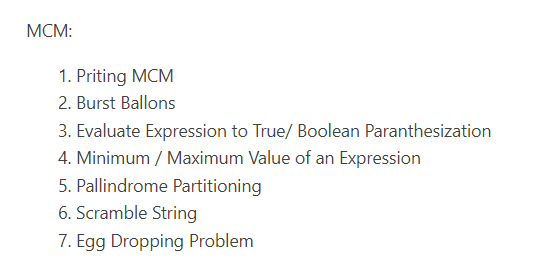
# Brackets/ expression/ stack dp

### 32. Longest Valid Parentheses

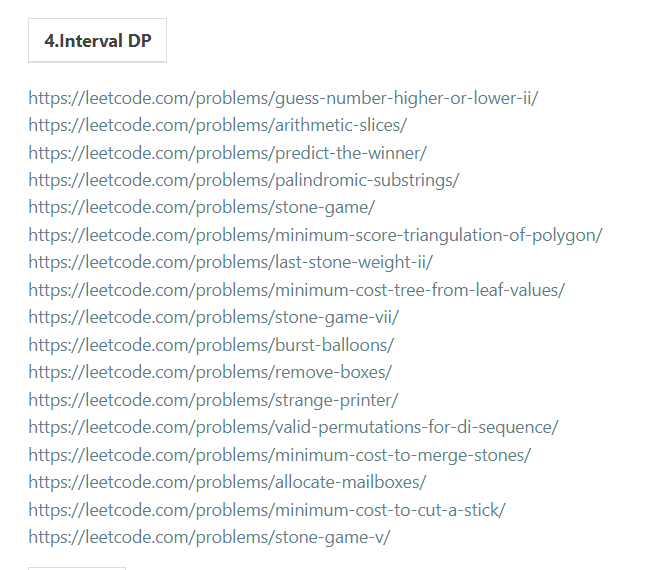


SOLN: done by codeNcode very famous

# MCM

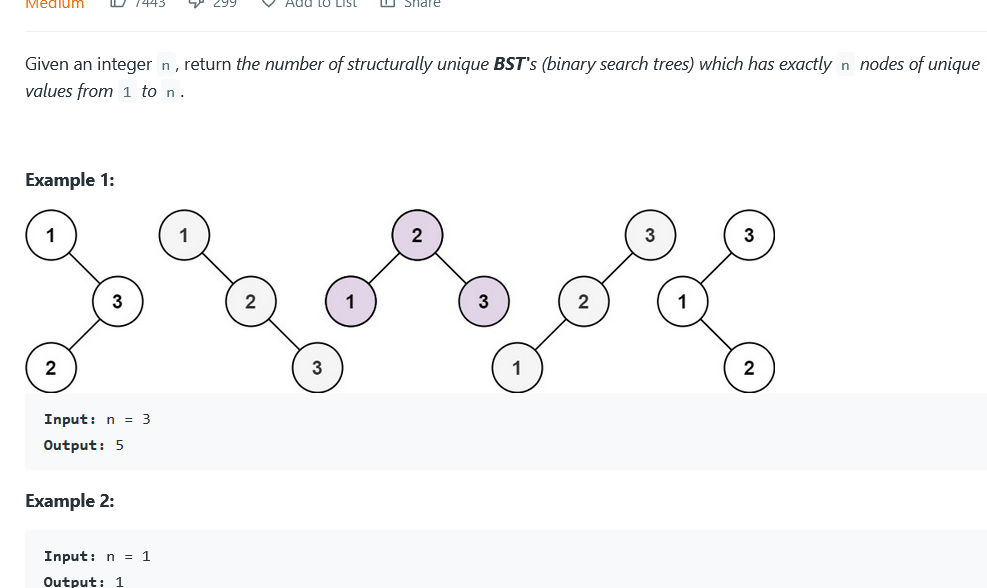


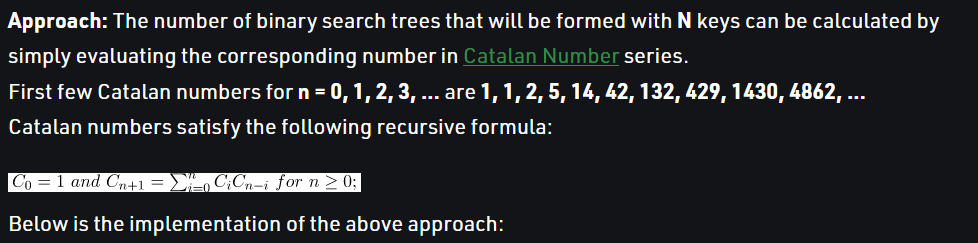
# Interval

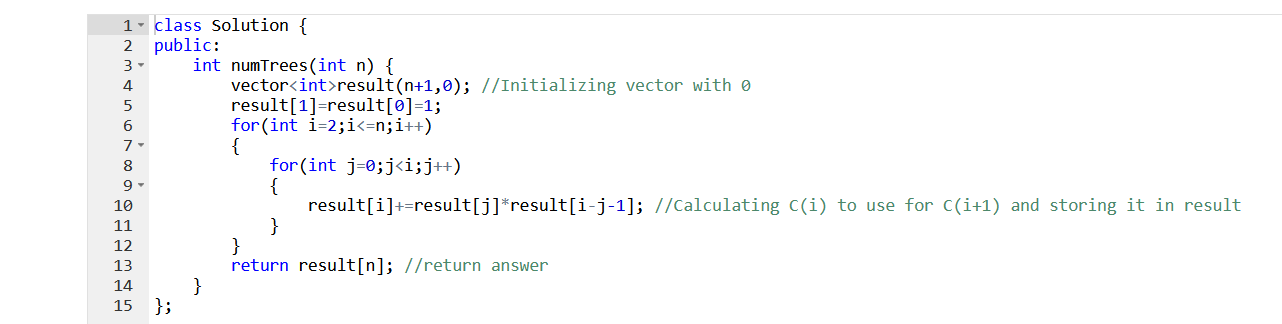


# MATHS

### Catalan number

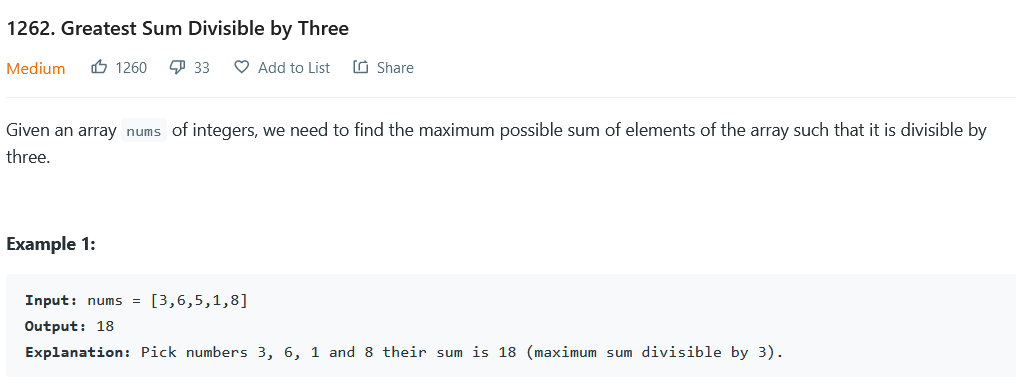




1 1 5 …

c( n) = sum of c(i)\*c(n-1-i) where i =0 to n-1

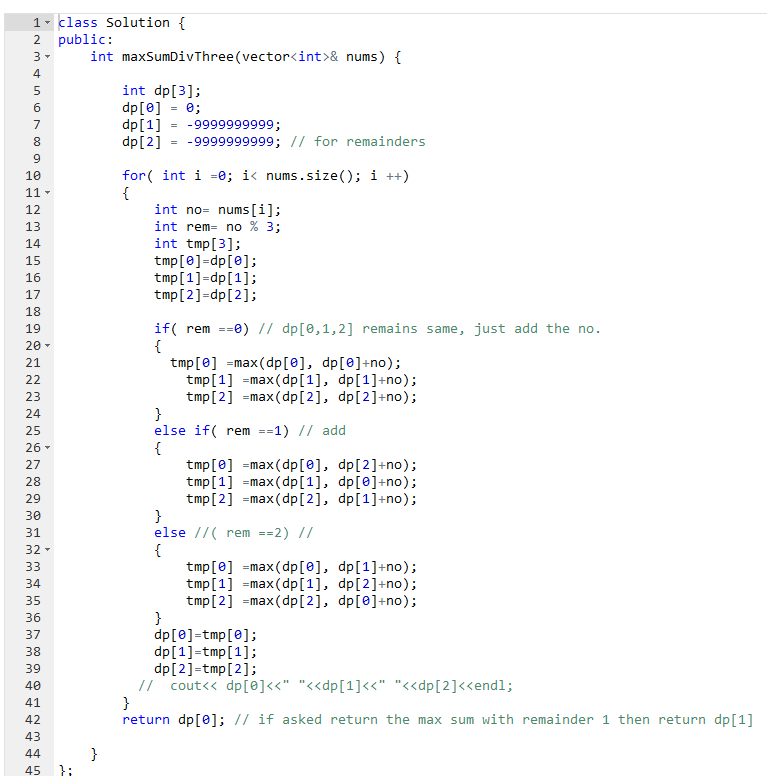
### Largest sum divisible by 3



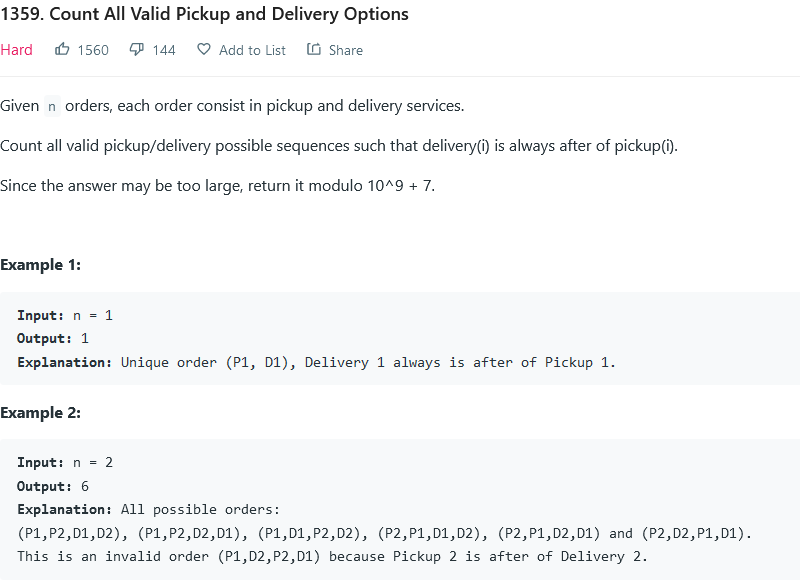
**Constraints:**

* 1 <= nums.length <= 4 \* 10^4
* 1 <= nums[i] <= 10^4

SOLN: <https://leetcode.com/submissions/detail/687851926/>



### Combinatorics



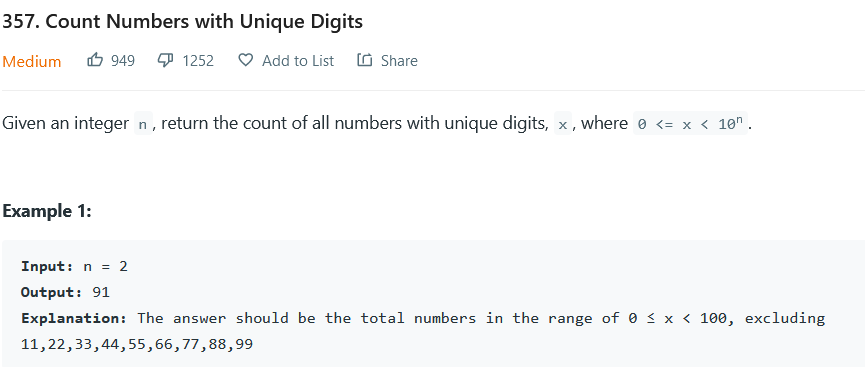
n=500

Soln:

<https://www.youtube.com/watch?v=p1tvA-eQFqk>

# Digit dp

### Leetcode medium

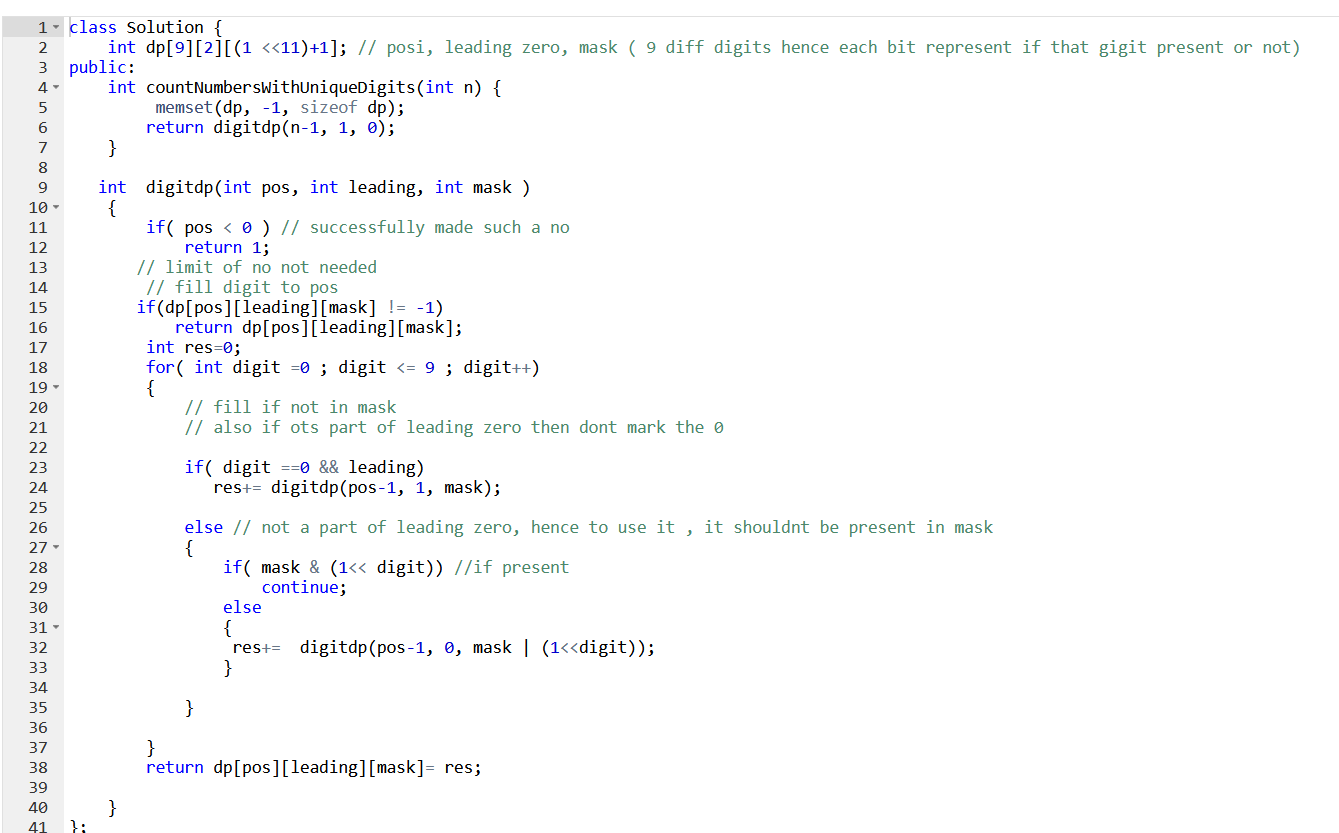


Constraint \

N =8

Soln:

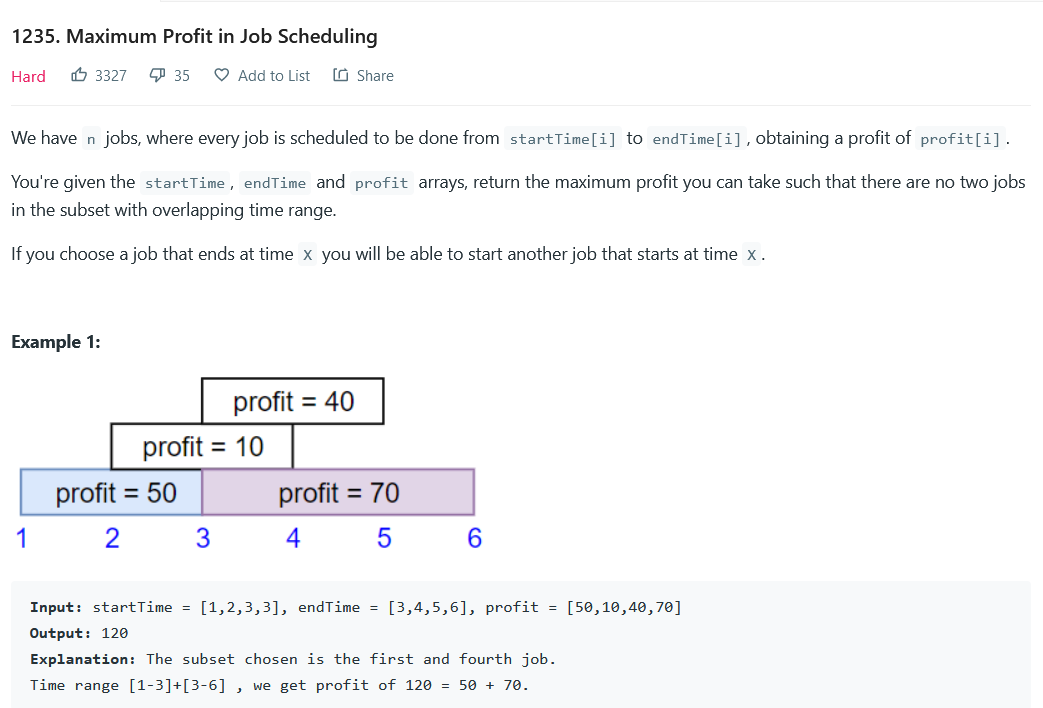
<https://leetcode.com/submissions/detail/687135583/>



# Type: Scheduling problem

Mostly are Np hard

### Job scheduling



**Constraints:**

* **1 <= startTime.length == endTime.length == profit.length <= 5 \* 104**
* **1 <= startTime[i] < endTime[i] <= 109**
* **1 <= profit[i] <= 104**

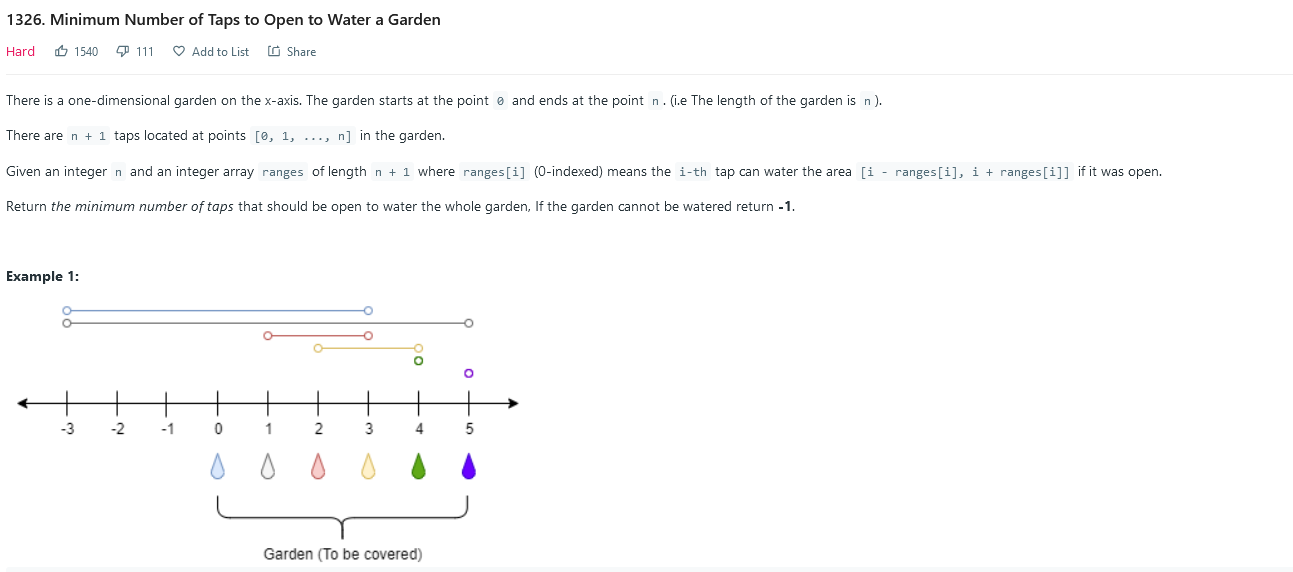
**SOLN: to take or not to take + binary search**

Sort acc to time. Always same with interval problems.

If a job isnt selected send to next job.

If selected send to a job whose starting time is >= ending time of current time

### (push dp)Minimum taps to water the garden



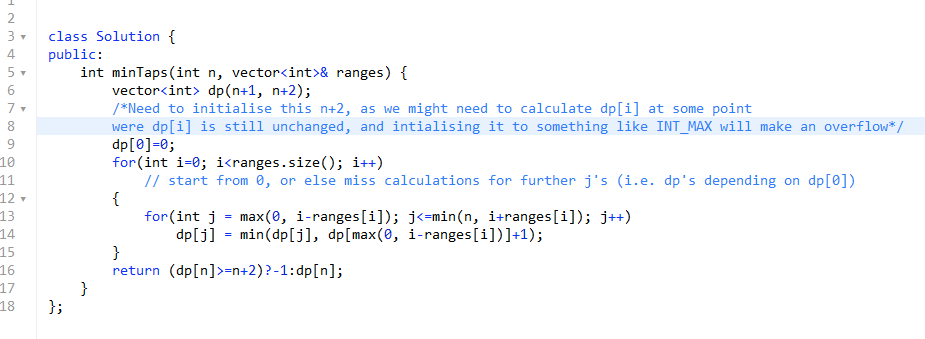
**Constraints:**

* 1 <= n <= 104
* ranges.length == n + 1
* 0 <= ranges[i] <= 100

**SOLN:**  dp[i]= min taps needed to water the garden till ith point.

N taps hence n+2 is equivalent to inf. Hence initializing with n+2

Its push dp.



# When to buy and sell Stocks

# Game theory

# Knapsack

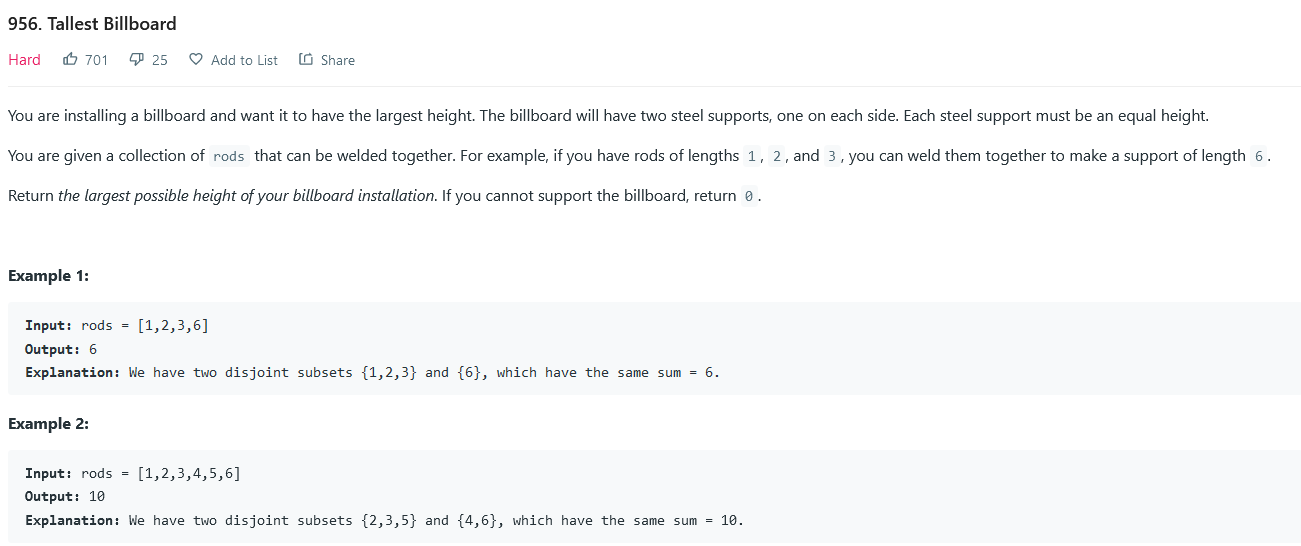
### Ques1

**Constraints:**

* cost.length == 9
* 1 <= cost[i], target <= 5000



### Ques 2 tallest billboard



**Constraints:**

* 1 <= rods.length <= 20
* 1 <= rods[i] <= 1000
* sum(rods[i]) <= 5000

SOLN: dp[left] [right] [i]

When rods finished check if left == right if yes return the ( left)

Complexity = left \* right\* i = 5000\* 5000\* 20

TLE

**AP2**

Smartness.

At last we have to check if left ==right i.e if left-right =0

Hence instead of storing left , right

Store left-right.

For any ith rod.

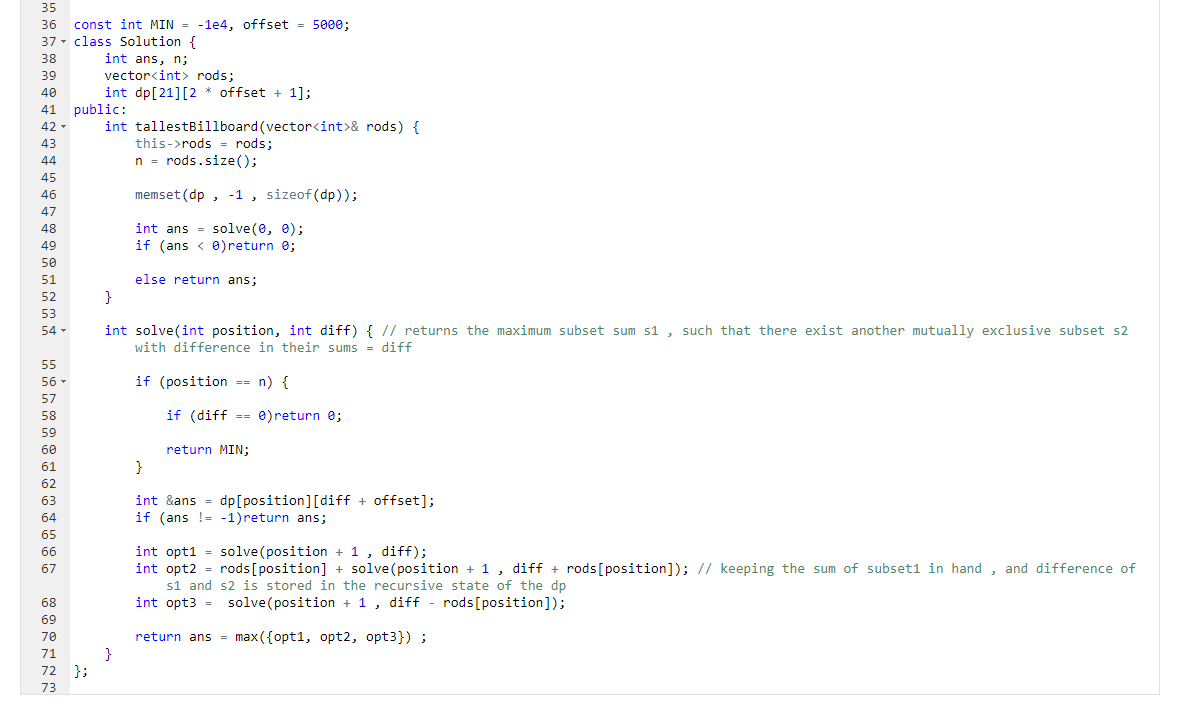
2 options, either add to left side i.e increase diff.

Or add to right height dec diff.

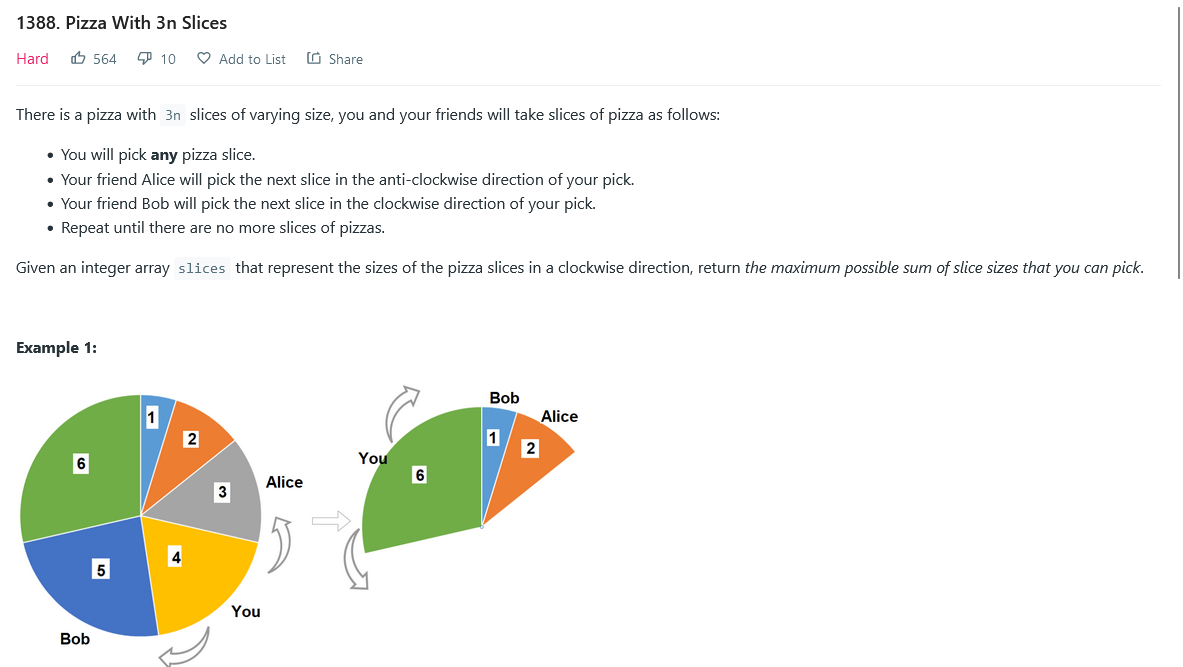
If at last diff =0 return 0

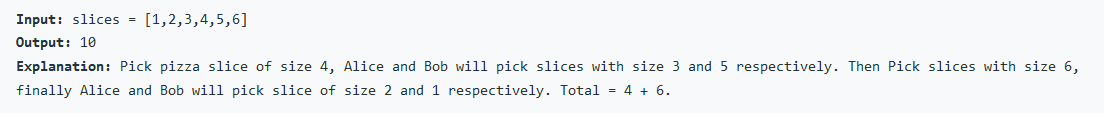
Else return -inf.

NOTE: when rod is added to left side then only length of billboard increases, else it doesnt matter( bec then the diff wont be 0)



### Circular knapsack 3n pizza





SOLN:

Same as house robber2. Robbing house in a circle

<https://leetcode.com/submissions/detail/689058402/>

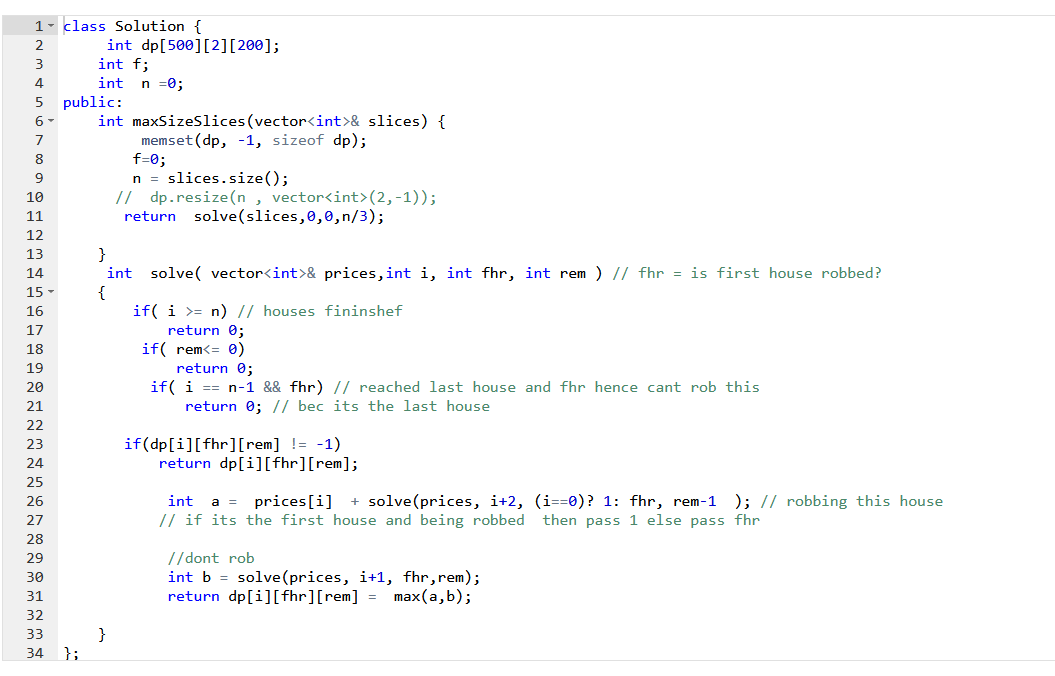
Soln:

Bec in every turen 3 pizzas are picked, hence only n/3 turns to pick pizza.

If first pizza is picked then lsat pizza disappears. But we dont have to worry abt that bec we are keeping track of how many pizza we can pick.

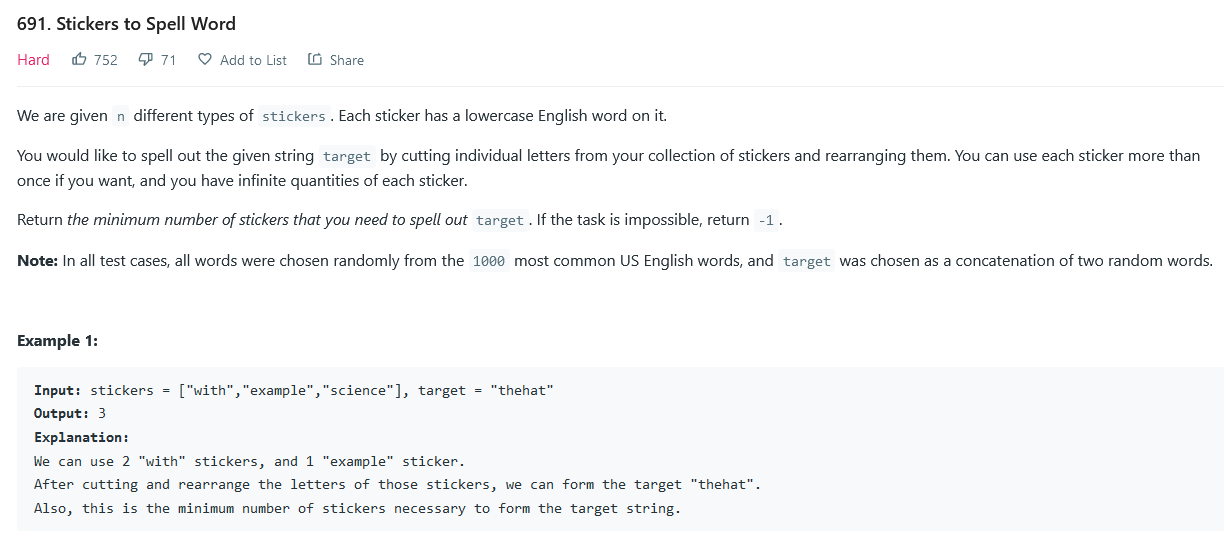
We wont worry to finish the whole pizza.3

The optimal ans will automatica;;y give us ans in which pizza gets finished



# Bitmask

### ques1



SOLN:

<https://leetcode.com/submissions/detail/730476368/>

Dp[mask] = min stickers required for this.

Cant use mask for 26 alphabets.

Use target string only 15 characters. Mask is from 0 to 14 character.

mask=2^15.= 32768

Make freq array for all stickers

vector<vector<int>> words(sticker.size)

For i in sticker.size()

vector<int> freq(26,0);

For j in sticker[i].size()

Freq[sticker[i][j] ] +=1;

words.push\_back(freq)

dp( mask)

` for all ith stickers( 50 stickers)

old\_mask= mask

Copy = arr[i] . copied freq array of ith sticker( 10 operation)

for( j =0 to 15) // mask ( 15 operation)

if( mask &(1<<j) ) // present in target

if(copy[target[j]-’a’] != 0) // present in sticker also

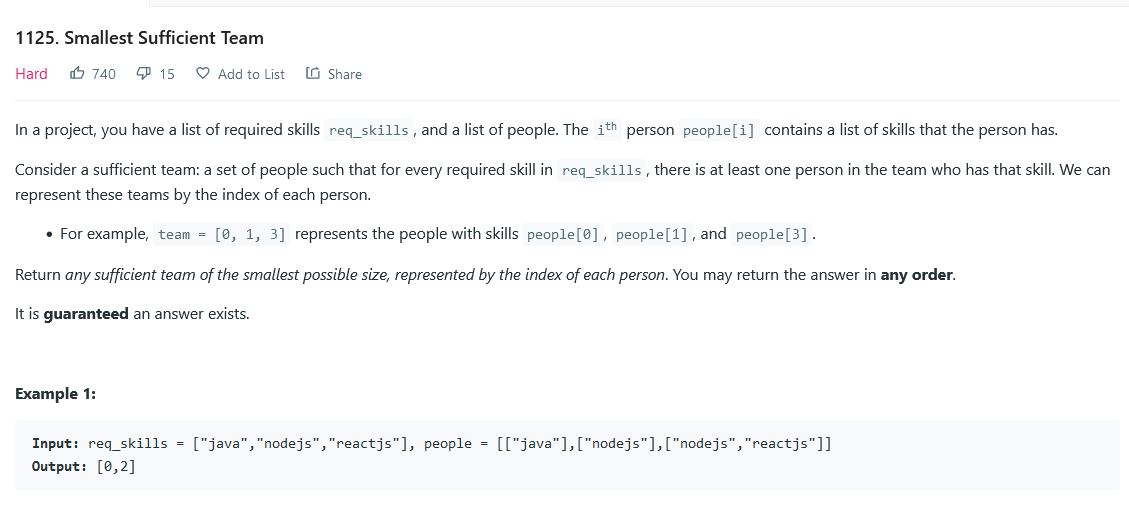
mask= mask^ ( 1<<j) // put that char done

if( old\_mask != mask) // i.e this sticker was useful .

ans= min(ans, dp(mask)

Return ans;

### Min set

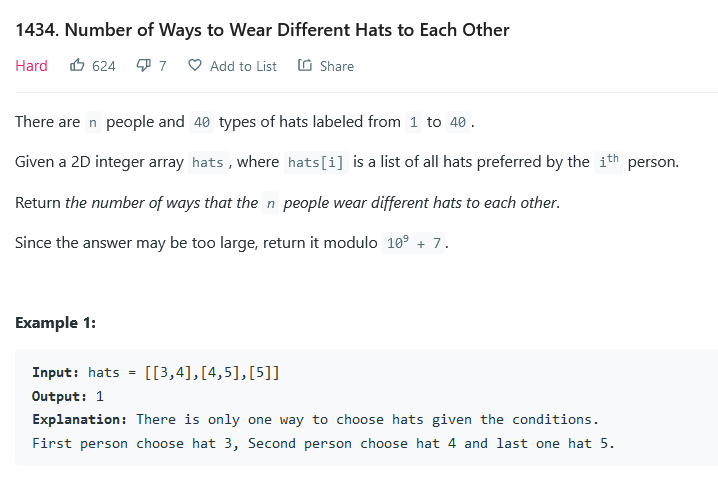


**Constraints:**

* 1 <= req\_skills.length <= 16
* 1 <= req\_skills[i].length <= 16
* req\_skills[i] consists of lowercase English letters.
* All the strings of req\_skills are **unique**.
* 1 <= people.length <= 60
* 0 <= people[i].length <= 16
* 1 <= people[i][j].length <= 16
* people[i][j] consists of lowercase English letters.
* All the strings of people[i] are **unique**.
* Every skill in people[i] is a skill in req\_skills.
* It is guaranteed a sufficient team exists.

<https://www.youtube.com/watch?v=yiAaGRWdqVA>

### Hat Assignment prob



* n == hats.length
* 1 <= n <= 10
* 1 <= hats[i].length <= 40
* 1 <= hats[i][j] <= 40
* hats[i] contains a list of **unique** integers.

SOLN:

People are less and hats are more.

Hence instead of assigning hats to people. Make people wear orr not wear a hat. I.e assign people to hat.

NOte; in this base cond is when all people are wearing hat, a successful one way , and when hats finished no way possible hence return 0.

For every hat there are 2 choices.

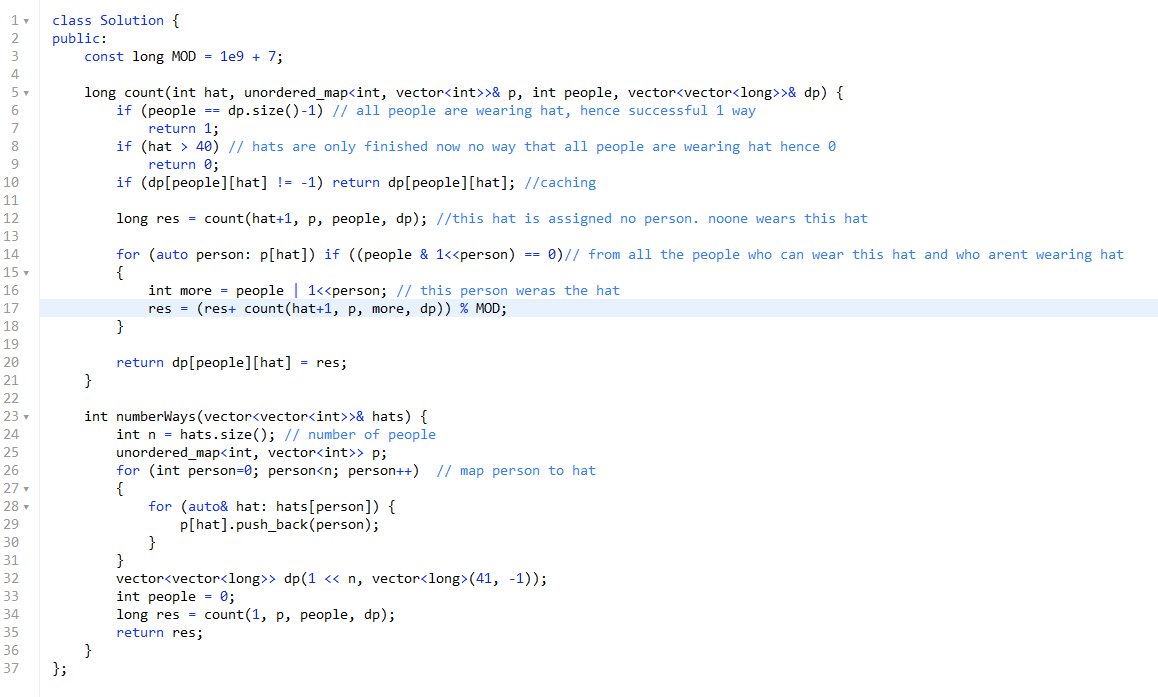
No one wears this hat

Someone wears this hat, who hass this hat and isnt wearing other hat.

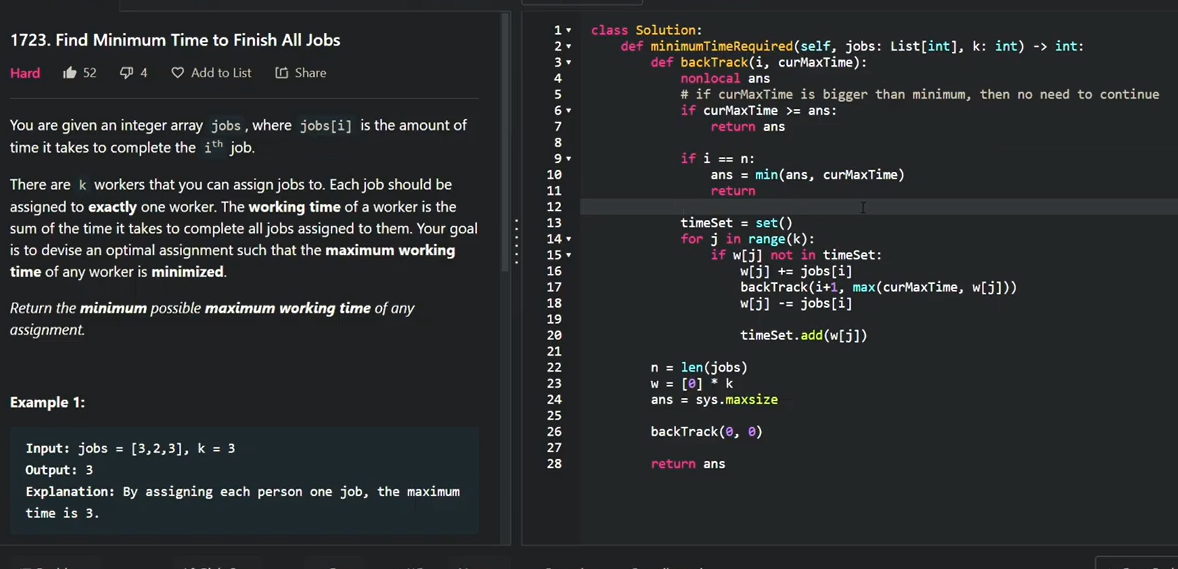
mask[1<< #people]

Dp[mask] [hat] = we are at ith hat and mask people have already worn hat, what is the no of ways we have?

dp( mask, i).



### Min job time pruning



**onstraints:**

* 1 <= k <= jobs.length <= 12
* 1 <= jobs[i] <= 107

**SOLN**

Timeset is to prune giving job to a worker which has same time as some prev one.

Why we can avoid this. Bec

Say we have workers [ 12,20,4,20]

Giving ith job to 2nd worker we got some ans.

Now giving ith job to 4th worker is same as giving to 2nd worker, will get same ans, hence can prune. And its imp else tle