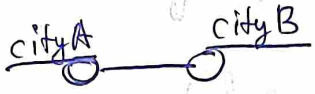
Codeforces Round #408 (DIV. 2).

(C) You will be given n cities, (n-1) edges represented in the form of a tree. There are K police stations placed on some cities out of the n cities.

The condition for a city to be safe is that that city should have atleast 1 police station at a distance K. Each edge has weight 1.

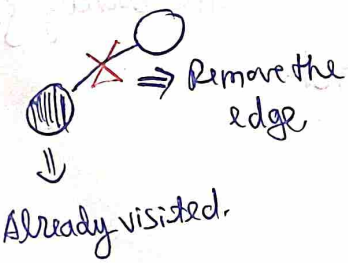


The city is made in such a way such that all the cities are already safe.

You need to cut the maximum number of edges such that each city is still safe. Even after cutting the edges

(A) Intuition- Since we know that all the cities are already safe. meaning each city has a police station at a distance K from it.

We can start doing bfs from each of the police stations. and whenever we will see that we are trying to visit an already visited cell, we will remove that branch.



(Proof Wikipedia) Bezout's Identity

Math function. This is a theorm that proves that
For two integers (a, b)
if $\gcd(a,b) = d$, then there exists integers x and y
such that $x \cdot a + y \cdot b = d$.

This can be extended to larger set of numbers. $(n >= 2)$
For two integers (a, b, c, d, e, f)
if $\gcd(a, b, c, d, e, f) = g$.
then $x_1 \cdot a + x_2 \cdot b + x_3 \cdot c + x_4 \cdot d + x_5 \cdot e + x_6 \cdot f = g$.
for some $(x_1, x_2, x_3, x_4, x_5, x_6)$.

Therefore if a question comes like,
Q) Given an array is it possible to use a subset of the array and multiply each of the elements with a separate multipicand, such that sum of those elements is g.

A) Basically. $\gcd(a,b,c) = g$. { For some subset (a,b,c) in the array }.
You need to find if any such subset exists or not.
Cuz if $\gcd(a,b,c) = g$
$a \cdot x_1 + b \cdot x_2 + c \cdot x_3 = g$ [Bezout's Identity].

Queue Reconstruction by Height.

Q) You will be given an array of people, People [hi, Ki],

hi - denoting the height of the ith people.

Ki - Number of people left of the person who are taller or of equal height.

Now you need to arrange the sequence in such a manner that the sequence is correct.

Input → [7,0], [4,4], [7,1], [5,0], [6,1], [5,2]

output → [5,0], [7,0], [5,2], [6,1], [4,4], [7,1].

Soln:

(Step1) → We will sort the data first on the basis of height and then on the basis of 2nd item.

So sorted list becomes like

[[4,4], [5,0], [5,2], [6,1], [7,0], [7,1]]

we have to fill 6 Blank spaces.

_ _ _ _ _ _

(Step2)

[4,4] → So 4 numbers are before [4,4], ~~#~~

[5,0] → No number before it → [5,0] _ _ _ _ [4,4] _

[5,2] → _ _ _ [4,4] _

Submask Enumeration

What is a submask of a bitmask?

Let say a bitmask is 1011000. Then its submasks are

1000000, 1010000, 1001000, 0011000, 0010000,

0001000,

0101000 → This is not a submask. The submask will have only those bits set which bits were originally set in the bitmask.

How to generate submasks of the bitmasks.

Code:

```
submask = bitmask;
while( submask != 0){
    submask = (submask - 1) & bitmask;
}
```

Suppose bitmask = 10 1100. Submask = 10 1100 as well.

(Submask - 1) ⇒ Will make the rightmast set bit 0 and make all the bits to the right of it as 1.

(101100 - 1) = (101011) & (101100) = (101000)

Now suppose ~~you~~ for each bitmask you want to generate their submasks. Lets try to find out its time complexity.

Lets say one of the bitmasks has K bits set. Now each of those K bits have 2^K choices. So 2^K submasks.

Now how many ways out of the n bits you can choose K bits.

$({}^nC_K \cdot 2^K)$. Now do this for all the K's from 0 to n.

$$\therefore \sum_{K=0}^{n} {}^nC_K \cdot 2^K = (2+1)^n = 3^n. \quad O(3^n)$$

Now Lets to a question on the above topic.

Amazon

Find Minimum Time to Finish All Jobs.

Leetcode

Q) You will be given an array, and you have to split them into K ~~subs~~ non-empty subsets, such that the subset which has the maximum sum is minimised. $O(K \cdot 3^n)$.

jobs = [1, 2, 4, 7, 8] K = 2

a) (1, 2, 8) (b) (4, 7) . Ans = 11.

A) We will be using Submask Enumeration in this question.

P.T.O.

(BITMASK Approach)

Lets say we have a dp[ ][ ] array.

dp[i][bitmask] ⇒ This will tell me that

For the first i workers, if we distribute

the jobs which are having their bits set in the bitmask,

then

dp[i][bitmask] = minimizing the maximum of all the subsets.

So we will build our dp in such a way that

dp[K][pow(2, numbr of jobs)] will be holding the final answer

n = no of jobs

(Step 1) Store the answer for 1 worker first

for(bitmask = 0 ; bitmask < (1<<n) ; bitmask++){

dp[0][bitmask] = sum[bitmask]

⇓

Sum of the jobs which have their bits set

(Step 2) Now while adding the ~~bitmask~~ a new worker we will use a submask.

Lets assume we have a bitmask 10101.

In the enumeration part we will be generating its submask.

So basically we will try to find.

Lets take a submask = 10001

$$dp[i][10101] = \min\Big(dp[i][10101],\ \max\big(dp[i-1][10001],\ sum[10101 - 10001] \big) \Big)$$

option1 ⇒ the submask divided among (i-1) workers

option2 ⇒ the rest of the jobs (i-1) (bitmask - submask) given to the i^{th} worker.

$$dp[i][bitmask] = \min\Big(dp[i][bitmask],\ \max\big(dp[i-1][submask],\ sum[bitmask - submask] \big) \Big);$$

Calculate this for all the bitmasks.

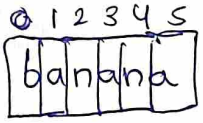
Suffix Array.

Construction and Utility.

What is Suffix Array?

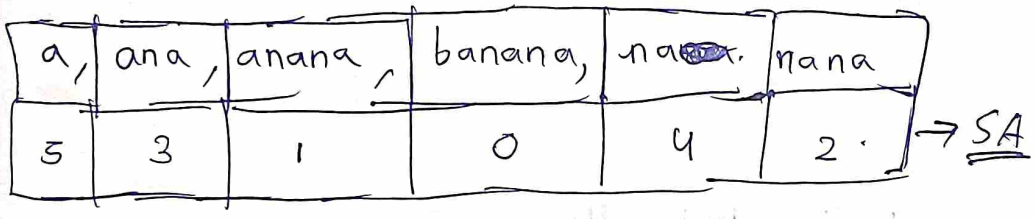
Let us say we have a string → banana.

Suffix array is an array which stores the suffixes in the sorted order.



→ Listing the suffixes. banana, anana, nana, ana, na, a.

If I sort them. it is stored as



Suffix Array stores the indices where the suffixes would have started. in sorted order.

Now there is a way using which we sort the suffixes.

We sort the whole suffix array in batches.

① First we sort the strings on the basis of 1st character.

② Then we sort it on the basis of first 2 characters

③ Then first 4 characters.

④ Then first 8 characters.

⑤ and then ... — first 2^n characters.

How do we do this?

Check out the next page.

Step 1) We create a p array. p[100][Len of string]

p[i][j] → will tell us what is the rank of the suffix j if all the suffixes are sorted by the first 2^i characters.

So first thing

```
for (i = 0; i < n; i++) {
    p[0][i] = s[i] - 'a';
}
```

(Step 2)

We will create a struct called suffix..

```
struct suffix {
    int first;
    int second;
    int index;
};
```

Q) You might be wondering why do we need first and second variable.

A) Because suppose we have p[i][j] → calculated for all the suffixes.. That means we have the suffixes sorted by the first 2^i characters.

Now suppose I want to ~~sort~~ calculate $p[i+1][j]$.

So now the first = $p[i][j]$, second = $p[i+pow(2,i)][j]$

Now if we take a pair (first, second). Then we have the suffix j, first 2^{i+1} characters.

```
i = 1;                                          (logN)
for(i=1; i<n; i++){                            ↗
    int next = pow(2, i-1);                    |
    if(next >= n) break;                       |
    for(j=0; j<n; j++){                        |
        t[j].first = p[i-1][j];                |
        t[j].second = j+next<n ? p[i-1][j+next] : -1;
        t[j].index = j;                        |
    }                                          |
}                                              ↘
```

Total is $O(N(\log N)^2)$

this sorting can be done in $O(N)$ or $O(N \log N)$

sort(t, t+n, comparator); ⇒ Just to sort the suffixes

Now we want to populate the p[][] array with ranks

```
for(j=0; j<n; j++){
    p[i][t[j].index] = (j>0 and t[j].first == t[j-1].first and
                        t[j].second == t[j-1].second) ?
                        p[i][t[j-1].index] : j;
}
```

this part is basically the condition of giving the same ranks to same suffixes.

Calculating the LCP → O(logN) complexity.

LCP → Longest Common Prefix

Given two suffixes, LCP will find what is the longest common prefix between these two suffixes.

Code:

```
int lcp(int i, int j, int n, int step) {
    if (i == j) return n - i;
    int return_value = 0;
    for (int K = step ; K >= 0; K--){
        if (i >= n or j >= n){
            break;
        }
        if(p[K][i] == p[K][j]){
            return_value += pow(2,K);
            i += pow(2,K);
            j += pow(2,K);
        }
    }
    return return_value;
}
```

step tells us till which point did we sort (i-e)

2^step is the last value using which we sorted suffixes

All characters are same.

Check which rank matches

that means 2^K characters are matching

shifting suffixes by 2^K characters and again compare

LEETCODE Weekly Contest MEDIUM

(Shortest String that contains all the three strings.)

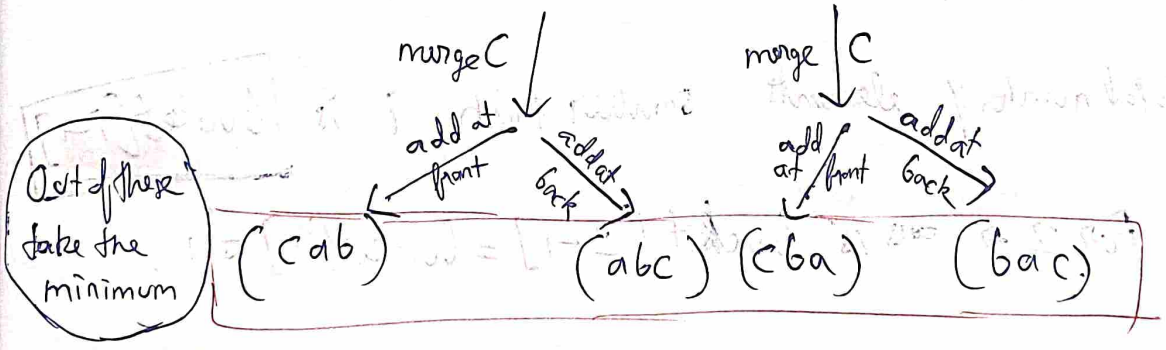
NEW APPROACH Completely

(Q) You will be given 3 strings a, b, c. Your task is to find the minimum ~~string~~ length string such that this string contains all the 3 strings (a, b, c) as substrings.

(A) We will have to brute force and manually combine the strings in all possible formats and get the minimum string out of those.

F(a, b, c) → In this first we will combine (a and b) and then try to merge c with it.

merge (a, b) → (a b) or b a



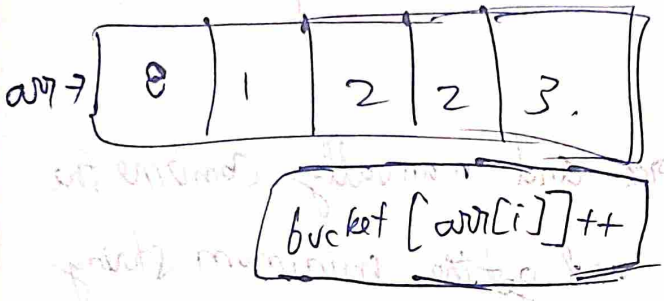
Similarly do for. F(b, c, a) → merge (b and c) and then try to merge a with it.

Finally take the minimum of. F(a, b, c), F(b, c, a), F(a, c, b)

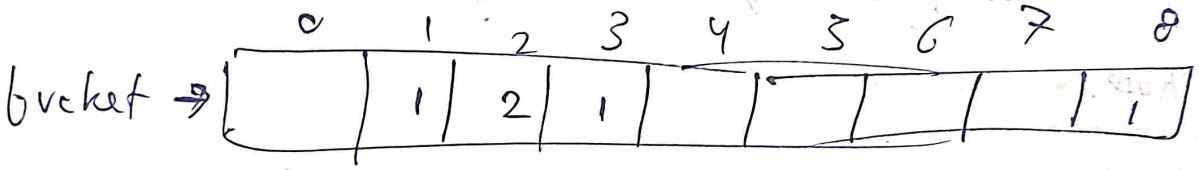
(LEETCODE) O(N) <Bucket Sort Algo>

Q1 <How many Numbers are smaller than the current Number>

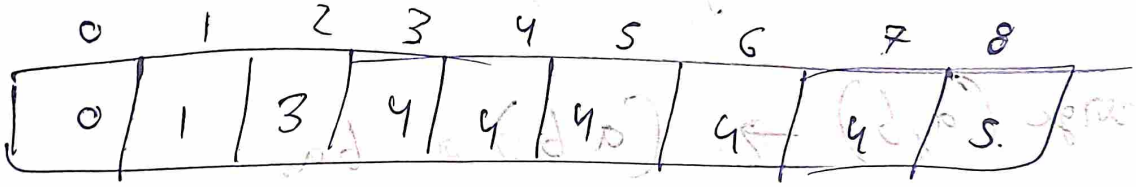
Given an arr, with $0 \le arr[i] \le 100$. For each number, tell the count of numbers which are smaller than it in the whole array.



We will use bucket sort which is an O(N) algorithm.



Now do a prefix sum on bucket



Now total number of elements smaller than i is bucket[i-1]

For 2 → ans is bucket[2-1] = bucket[1] = 1.

(LEETCODE) (CONTEST) (**) Check If It is Possible to Split Array.

You are given an array of length n. and an integer m. You need to be able to tell if it is possible to split the array into n, non-empty arrays.

In each step for splitting the array, the following conditions must hold.

* Length of the subarray is 1 or.
* The (sum of elements of the subarray) is $\geq m$

(Soln) Method 1: $O(N^2)$

Solve this using dynamic programming.

dp[i][j] → will basically tell you if you can split the subarray arr[i, i+1, ..., j].

Inside dp(i,j) → you can perform all possible splits, and break it into subproblems. dp(i,j) → dp(i,k) && dp(k,j).

(Method 2) (**) [Amazing Observation] $O(N)$

If any two consecutive elements have sum $\geq m$. then I can split the entire array into single element.

(LEETCODE) (CONTEST) **Find the Safest Path in the Grid** (SEE CODE) (Imp.) (DISCUSS) (★ ★ ★)

You are given a N×N matrix. A cell which is empty grid[i][j] = 0 and a cell which is a theif has grid[i][j] = 1.

For a given cell, the safeness factor of the cell is the minimum manhatten distance from any of the theif cells.

After we know the safeness factor of each cell. We need to ~~construct~~ find the path such that in that path the minimum safeness is maximum. ~~You~~ Out of All the paths, you need to choose a path from (0,0) to (n-1, n-1) where the minimum safeness is maximum.

Soln

Step1) You can find the safeness of each cell using Multi-Source BFS. Now for each cell you have safeness factor.

Step2

(Method 1) (Binary Search + BFS) ($\log N$) * N^2

(BFS, DFS)

In this we will do a binary search.

choose a mid = (start+end)/2. And we will check if there's a path that exists such that any cell we travel has (Safeness ≥ mid). And manipulate our mid.

(Method 2) ($\log N^2$) (N^2)

Priority Queue based Dijkstra

We insert into the cell,

{Safeness of the cell, index of the cell}

(LEETCODE CONTEST) (Length of Longest Valid Substring) (HARD)

(SLIDING WINDOW)

(CODE IN LEETCODE)

You are given a string word and also given a set of strings → (Forbidden Strings). A string is called Valid if non-of its substrings are a part of forbidden Set.

We need to print the length of the longest valid substring of ~~which is a part of~~ word.

Example word = [c|b|a|a|a|b|c] → valid substring

forbidden = ["aaa", "cb"]

(Soln) We will be using a sliding window approach on solving the problem.

Observation ① → You see that the maximum length of a forbidden String is 10. → (V.Imp Observation.)

Observation ② → [Active Window] + (new character).

All the new substrings generated will decide whether this new character can be included in the Active Window.

Now we only care about the substrings generated of length 10.

newcharacter, [window | newcharacter], ... [Active window (9 characters) | new character]

← length 10 →

(Then Repeat) (Till Window goes out of String)

(LastStep) → If the new set of 10 substrings are present in the forbidden Set. Then the window has to be changed.

Suppose.

(Forbidden String)

[| i | new character.] → [| i | i+1 | new character]

New Active Window

← Active Window →

(Premium)

(Amazon) **Find the Index of Larger Integer** (★★★)

$O(\log_2 N) > O(\log_3 N)$

You are given an integer Array, where all the values are same except that 1 Integer. A single index has a value larger than all the other indexes. You are given a function (★★★★)

there's a follow up.
1. what if 2 integers are there.
2. what if 1 is smaller and 1 is Larger.

$f(l, r, x, y)$ → which tells us.

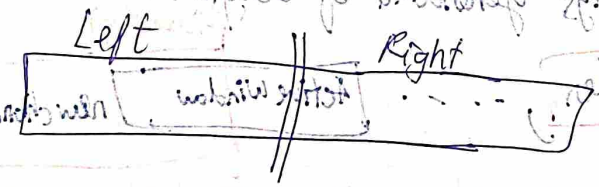
{
sum(arr[l] + ... + arr[r]) comparison with
sum(arr[x] + ... + arr[y])
}

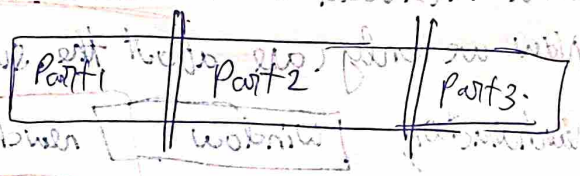
So $f(l, r, x, y)$ returns 1 → if subarray sum is greater,
returns −1 → if subarray sum is smaller, 0 → if subaray sum is equal.

(Soln) ~~Brute~~ **Binary Search** $(\log_2 N)$ || V. Imp (Cool Trick) **Ternary Search** $(\log_3 N)$

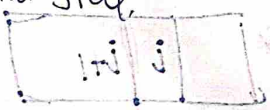
could use two parts of the array-

Divide the given array into 3 parts with





Using function $f(Left, Right)$ which ever has Larger sum, go to that side.



$f(Part1, Part2)$

if (Part1 = Part2) then search in Part3
if (Part1 > Part2) search in Part1
if (Part2 > Part1) search in Part2.

(V. Imp) (#) Meeting Scheduler. (Interval Problem)

(Leetcode Premium) (TIK TOK)

You are given 2 arrays. Slots1 and Slots2. and an Integer Duration You need to find the earliest slot available in which both are available.

Slots1 → [[10,50], [60,120], [140,210]], slots2 → [[0,15], [60,70]]

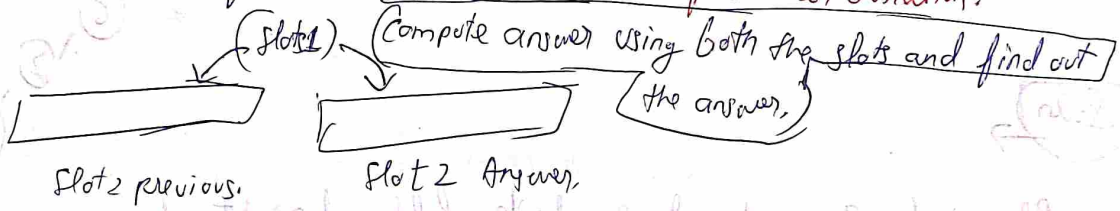
duration → 8.

Ans → [60,68]

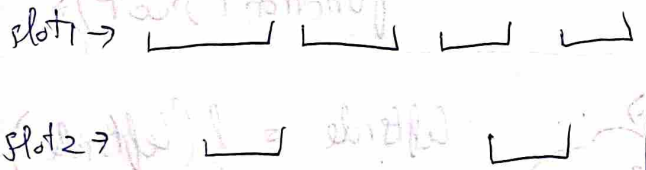
(Solution 1) → (Binary Search)

Sort (Slots1, slots2). After sorting them loop through each slot of (Slots1) and using Binary search find the slot in Slots2 whose start is just greater than or equal to the start of the current slot of Slots1.

Corner Case → Filter out slots which are of smaller duration.



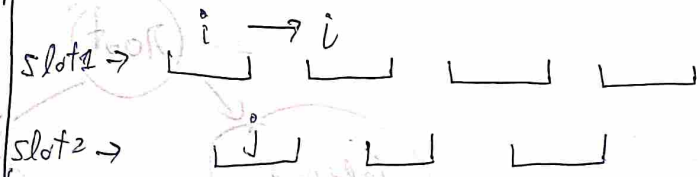
(Solution 2) → (Priority Queue)



Since slots of same person wont collide So we use a priority queue.

check for intersection between slot i and slot j and pop slot i

(Two Pointer)



We keep i and j and check their compatibility. Out of i and j whichever slot has earlier ending time we move it forward. In this case i

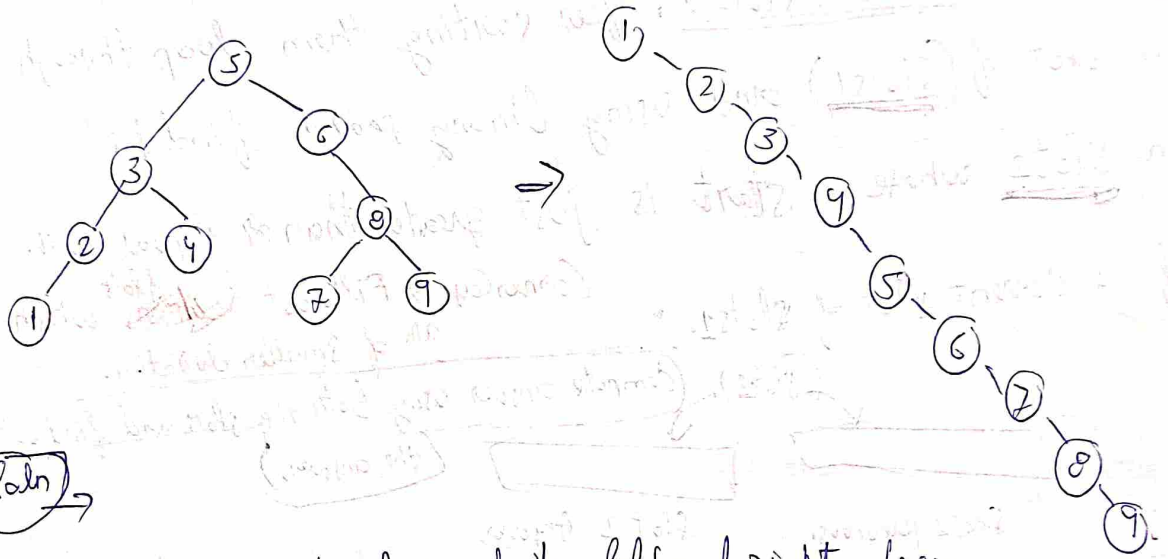
3.

Follow up If there are multiple people with multiple slots. Then calculate the same. If not all are available at the same time. Find the slot in which maximum people are available.

[Line Sweep we can use that] → then check for any duration, which duration has maximum people using subway pepisson.

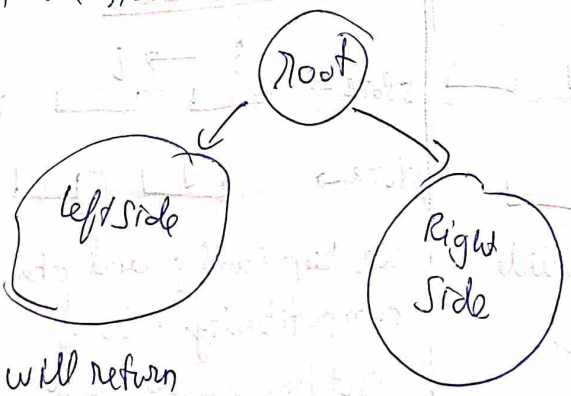
(GOOGLE) Increasing Order Search Tree

You are given a binary search tree, rearrange the tree in in-order, so that the leftmost node in the tree becomes the root of the tree and every node has no left child and only right child.



Soln →

- Plan is to segment and evaluate left and right trees seperately and merge with the root




```
function (root){
    LeftSide = f(LeftSide)
```

f(LeftSide) will return Head and tail of the chain of LeftSubtree. we want to merge

[tail of LeftSubtree chain]
LeftSide.second → right = root
root → left = NULL
root → right = RightSide.first [Head of RightSubtree chain]

f(RightSide) will return the Head and tail of the chain of RightSubtree.

Code → Any = f(root).first.

```
pair<TreeNode*, TreeNode*> f(TreeNode* root){
    if(root == NULL){
        return {NULL, NULL};
    }
    pair<TreeNode*, TreeNode*> leftSide = f(root->left);
    if(leftSide.second != NULL){
        leftSide.second->right = root;
    }
    root->left = NULL;
    pair<TreeNode*, TreeNode*> rightSide = f(root->right);
    root->right = rightSide.first
    TreeNode* head = root;
    TreeNode* tail = root;
    if(leftSide.first != NULL){ head = leftSide.first; }
    if(rightSide.second != NULL){ tail = rightSide.second; }
    return {head, tail};
};
```

Merging

⇒ leftSide Processing.

⇒ Right Side Processing.

(GOOGLE) [Robot Room Cleaner]

Q) You have been given a robot and you have to use that robot to clean the room. The room has some of its cells blocked, and some cells are empty. ~~Your job is to say~~. You have been given the following APIs now you have to write an algo such that the blindfold robot cleans the entire room.

Robot → {

turnRight(); → turns the Robot to right by 90°.

turnLeft(); → turns the robot to left by 90°.

move(); ⇒ Move in the direction by 1 cell it is facing. returns false if there is a block, true if there isn't

clean(); → Cleans the current cell in which the robot is present.

}.

Soln 1) Idea is to solve it using dfs and backtracking. So basically we will traverse through the grid and check for all possible directions the Robot can travel and we will consequently clean the cell in which we are in.

2) The starting cell we will mark it as (0,0) and the remaining cells we will mark it relative to this cell.

3) Upon reaching a cell where all the options are blocked or we have already explored all the options we will backtrack to where we originally came from / [basically to caller method] with

the same orientation. For that we will do the following operation.

turnRight() x 2 times → 180° to go in reverse.

move() → x1 time

turnRight() x 2 times → 180° to go regain original orientation.

directions → [{1,0}, {0,-1}, (-1,0), (0,1)]
Right, Down, Left, Upper

set<pair<int,int>> visited;

```
dfs( int x, int y, int current_orientation){
  visited[{x,y}] = 1
  clean();

  for(int i = 1; i <= 4; i++){
    int new_orientation = (current_orientation + i) % 4;
    int newX = x + directions[neworientation].first;
    int newY = y + directions[neworientation].second;
    turnRight();
    if( !visited[newX][newY] and move()){   // ⇒ cell is empty
      dfs(newX, newY, neworientation);
    }
  }
}
```

```
turnRight();
turnRight();
move();
turnRight();
turnRight();
```
⇒ Explained in point 3)

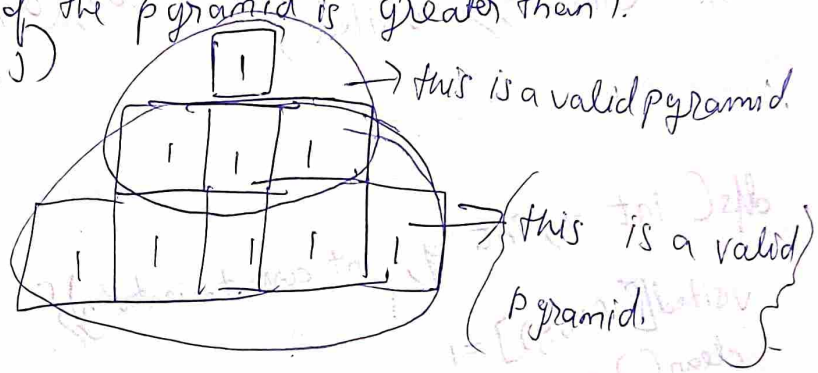
Explanation.

Let's say
current orientation = 3.
i.e. Upper.
Robot is facing Up.
If we do
(Up+1 = 3+1 = 4) % 4
= 0.
to it becomes Right.
So each time we are adding i we are basically turning Right()

(GOOGLE) ✓ Count Fertile Pyramids in a Land.

You will be given a n×m grid, where some cells will be 0 and some cells will be 1. You need to count number of pyramids and inverse pyramids that can be formed.

what is a pyramid. Lets take a cell (i,j). It is considered a pyramid if the height of the pyramid is greater than 1.

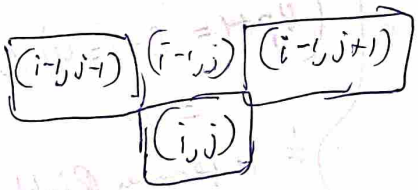


Similarly, inverse you can visualize

Soln

* This is a very simple problem. We will evaluate considering inverse pyramid formation.

Lets say a cell (i,j) will form a pyramid or not. and if it will how many will it form,



* Condition 1 → if (grid[i][j] == 1 and grid[i-1][j] == 1)
to ensure if height 2 can be reached.

* Condition 2 → Now we have already evaluated results for (grid[i-1][j-1] and grid[i-1][j+1]) so we know what height of Pyramids those are forming.

* (Step 3) → out of grid[i-1][j-1] and grid[i-1][j+1] minimum of them will be new height of pyramid.

grid[i][j] = min(grid[i-1][j-1], grid[i-1][j+1]) + 1.

Step 4 ans += (grid[i][j] - 1) { because number of pyramids with newly introduced will be height-1, since height=1 is invalid }.

[Code in LeetCode] | (GOOGLE)

[Code in LeetCode] [Range Module]. (HARD).

(***) You are fucked if this question comes in interview.]

Q) A half open interval concept is used here. [left, right) → means all real numbers within the interval of ≥ left and less than right.

Now you are given 3 operations.

1). addRange (int left, int Right) → Basically you need to add a range [left, right) in the set of valid intervals.

2) queryRange (int left, int Right) → (Self Explanatory) you have to tell all the real numbers within the interval [left, Right) are valid or not.

3) removeRange (int left, int Right) → You have to mark the range of real numbers between [left, Right) as invalid.

Sohn Yes I had the correct intuition behind it;

* Well for adding a range, we want to check if the new range is overlapping with any of the existing ranges, and if it does we want to merge it into a whole new range and remove the old Range. Basic idea is to have disjoint interval.

To begin with we will have 4 data structures.

startMap → Mapping of s→e

endMap → Mapping of e→s

startSet → all the entries of start

endSet → all the entries of end.

addRange (start, end){

* Step1) Find the end of ath interval which is just greater than or equal to start.

auto it = endSet.lower_bound(start);

Step 2) iterate over the endSet map to traverse through intervals.

while (it != endSet.end()){

int e = *it;

int s = endMap[e]; ⇒ potential overlapping interval,

it++ → (move pointer to next interval to process it during next iteration)

if (overlapping){

removeMapping (s, e); ⇒ mark the current interval as invalid.

}

else {

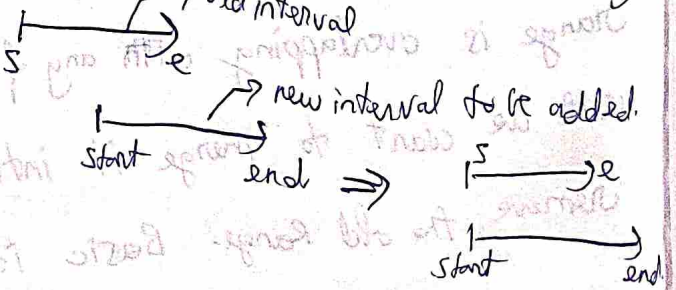
break; ⇒ Since (e ≥ start and if no overlap then rest of the intervals are also greater and non-overlapping

}

}

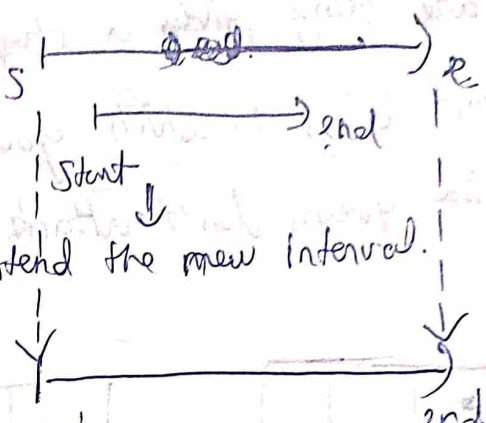
If (start > s){

start = s; ⇒ For the case when.



```
if (e > end){
    end = e;
}.
```

For the case when:



then extend the new interval.

} ⇒ close while loop.

Step3 addMapping (start, end) ⇒ add the new interval as valid.

removeRange (start, end){

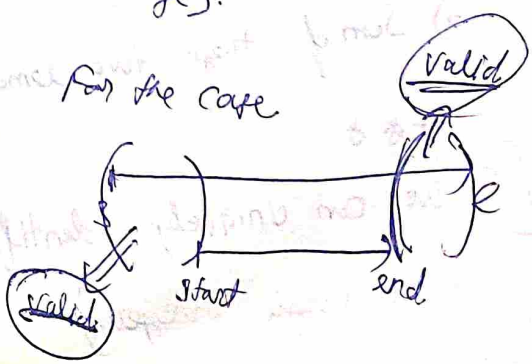
Step1 → Find the end of an interval which is strictly greater than start

Notice why I used strictly greater, because end = start is already invalid, because its a half open interval [start, end).

```
auto it = endSet.upper_bound(start);
while (it != endSet.end()){ ⟹ traversing through the intervals.
    int e = *it;
    int s = endMap[e];
    it++;
    if (overlapping){
        removeMapping (s, e) → mark it as invalid,
    } else { break; → {same reason as mentioned in addRange}.
    }.
    if (start > s){ addRange (s, start); }
    if (e > end){ addRange (end, e); }
}
```

⇒ close while loop

For the case

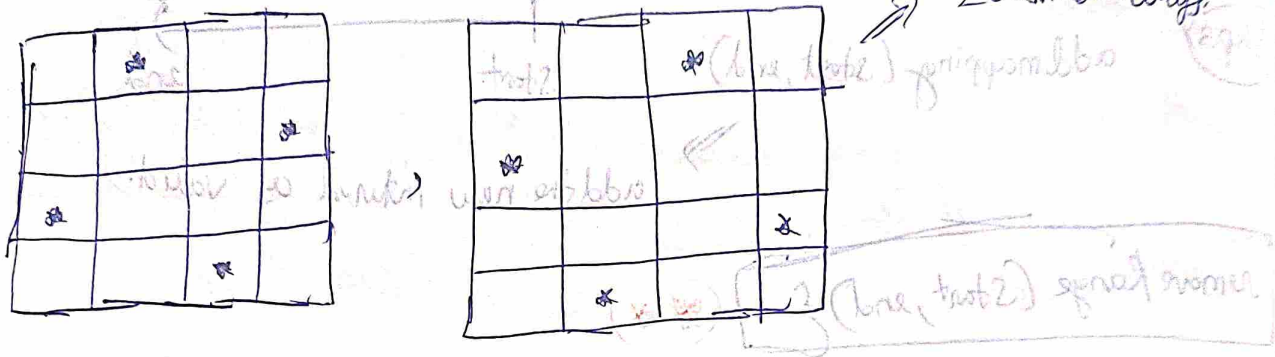


(GOOGLE)

Important New Concept → (N Queens - II) → [Code in Leet Code]

Q) You are being given a chess board, You have to find number (N*N) of distinct ways in which you can place N queens in such a way that the queens don't attack each other.

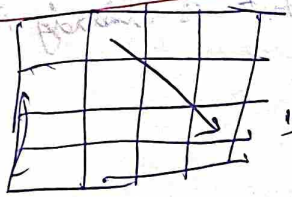
Example.



→ 2 distinct ways.

Soln Here I will introduce two important concept of a grid

Negative diagonal → if we are travelling in the direction from top left to bottom right



1) Then we can see at the next cell,

$row_{next} = row + 1$ and $col_{next} = col + 1$.

2) Difference between row and col remains same.

$diff = (row - col)$

Positive diagonal → if we are travel in the direction from top right to bottom right.

1) Then we can see at the next cell. $row_{next} = row + 1$ and $col_{next} = col - 1$.

2) Sum of these two remains same ; $Sum = (row + col)$

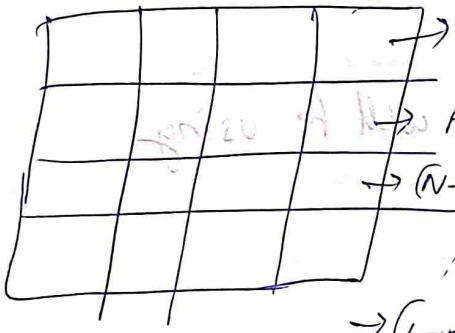
[So we can uniquely identify a neg-diagonal and positive diagonal using a single integer.]

Logic

We will use backtrack, and at each row we will be placing queens at each column. and check for 3 things

1). if this column falls into the visited column list.
2) if this cell falls into negative diagonal visited list.
3) if this cell falls into positive diagonal visited list.

Complexity Analysis. ⇒ V.V.Imp ⭐⭐⭐



→ Here we can try N different options
→ Here (N-1) options.
→ (N-2) ;
→ For each option we are going over N colums.
→ (1 options)

$(N \cdot (N-1) \cdot (N-2) \ldots 1)$ [N options] → $(N! \cdot N)$

GOOGLE

[Minimum Replacements to Sort the Array]

→ You already know the question. The crux of the question is, given a number X, Y such that $(X > Y)$. Split X into various parts such that [sum of the parts = Y] and $\min(\text{parts}_i)$ is maximised.

Soln ⭐⭐⭐ (V.Imp Part) Let's say $\{X = 25, Y = 12\}$. (Case I) $(X \% Y == 0)$ then $\min(\text{part}_i) = Y$

(Case II) If its not divisible: (Step 1) → Get how many parts it needs to be broken into such that each part is $\leq Y$. [$\text{part}_i \leq Y$]. After operating

$\text{Break} = (X/Y) + 1$. [The +1 is for $X \% Y$ part].

(Step 2) → Now check what is the maximum number you can fill in those Breaks.

Real $(\text{Answer}) \times (\text{Breaks}) \leq X \rightarrow (\text{Answer}) \leq (X/\text{Breaks})$

Imp

[Maximum Score of Good Subarray]

O(N)

Two Pointer

GOOGLE

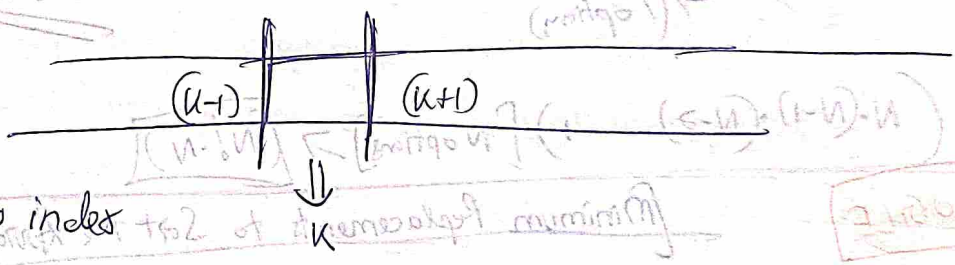
Q) You are given an array of integers and a number K. You need to find the maximum possible score of a valid subarray

Sol) For a subarray [nums[i], nums[i+1], ---, nums[j]]

Score = min (nums[i], nums[i+1], ---, nums[j]) * [Length of the subarray]

valid subarray (i,j) such that $(i \le K \le j)$ → i.e index K has to be included always.

A) The soln is really simple we will be using two pointers to solve this.



[Point 1]
out of the two index whichever we are going to choose length of the new subarray is going to be same in both the cases.

So out of nums[K-1] and nums[K+1] its better to choose the larger one

[Point 2] → whether to expand on the left or the right totally depends on which side has greater value, so that it maximizes the minimum of the new subarray

[Code in LeetCode]

(Burst Balloons)

Dynamic Programming | Amazing Problem | (Code In LeetCode) | (GOOGLE)

Q.) You are given n balloons, indexed from 0 to (n-1). Each balloon has number written on it.

If you burst ith balloon, we will get a few coins

coins = nums[i-1] * nums[i] * nums[i+1]. Basically the left & Right adjacent coins. But if there are no coins present to the adjacent of it. We take coin values to be 1.

Find the maximum points you can achieve after burst all balloons.

(A) (Step1) Lets break the problem into sub problem.

if we choose a sub array [i, ... , j]

and we want to burst the Kth balloon.

So our problem breaks into

Solve(i, j) = Solve(i, K-1) + leftballoon * nums[K] * Rightballoon + Solve(K+1, j);

[Problem of Bursting Kth balloon First]

leftBalloon and Rightballoon → should exist Right,

But we cannot burst the Kth balloon first and then solve the subproblem (i, K-1) because

[balloon[K-1] and balloon[K+1]] ⟹ are adjacent now

(How to solve this Problem)

(*) Lets say we will burst the k^{th} balloon at the last of the subarray $[i, \dots, j]$.

(*) So now $(k-1)^{th}$ and $(k+1)^{th}$ are not adjacent now.

(*) So now we solve the subproblem

1) solve (i, k-1)
2) solve (k+1, j)
3) (burst the kth balloon of the subarray at the last)

Picking the k^{th} balloon at the end will be the catch.

So leftBalloon = nums[i-1]
So rightBalloon = nums[j+1].

solve(i, j) = solve(i, k-1) (Left Subarray) + solve(k+1, j) (Right Subarray) + (nums[i-1] * nums[k] * nums[j+1])

(GOOGLE) [CODE IN LEETCODE] (24 Game) [Backtracking]

Q) You are given 4 cards with values in it. You should arrange the numbers on these cards in a mathematical expression using the operators ['+', '-', '*', '/'] to get the value target

Division operator is not integer division. its real division.

$3/2 != 1$, $3/2 = 1.5$

Sol: We will be using complete backtracking and complete brute force. Since we are allowed to arrange the numbers hence we need not apply operations just on adjacent numbers.

[Implementation is very nice. I was not able to implement. Pls Revise]

Concept of epsilon introduced for double numbers → epsilon = pow(10, -5).
↳ to compare double variables.

```
void backtrack( vector<double> nums){
    if (nums.size() == 1){
        if (abs(nums[0] - target) < epsilon){
            ans = true;
        }
        return;
    }
    for (i=0; i<n; i++){
        for (j=0; j<n; j++){
            if (i==j) { continue; }
            double a = nums[i];
            double b = nums[j];
            Vector<double> newnums;
            for ( k=0; k<n; k++){ if (k==i or k==j){ continue; }
                go.push_back ( nums[k]);
            }
            go.push_back ( a (operation) b);
            backtrack(go);
            go.pop_back();
        }
    }
}
```

[As we keep on modifying the array, for each operation performed on the two operands we shrink them to 1 operand]

here we choose which (i and j) to perform operation on.

we populate all the elements except i and j

we push the new operand at last

(Count Pairs with XOR in Range)

(GOOGLE) (LEETCODE) O(N)

(CODE KN LEETCODE)

Q) Given an array, and two integers low and high, return the number of nice pairs. A nice pair (i, j) of an array is such that $low \leq nums[i] \wedge nums[j] \leq high$.

(A) This problem is similar to the problem of find maximum XOR of 2 elements in the array

Break the problem into 2 parts-

① (a) Populate the numbers nums[0] ---- to nums[i-1] into the trie, using bits from left to right. Suppose 10 → 1010 →

(b) And at each node we have a cnt pointer to keep track that how many numbers have this bitwise prefix

② For each nums[i], we can traverse the trie and try to check number of elements inserted into the trie whose XOR with nums[i] is less than a given number K.

Imp part

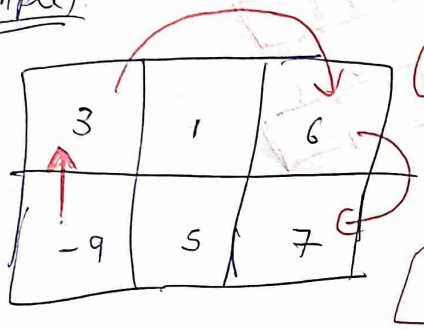
```
for (each bit of K) {
    ith bit of N = (N >> i) & 1;
    if (ith bit of K == 1) then we can consider all the numbers whose
        ith bit is same as that of ith bit of N - Because that
        will make XOR = 0 which is less than K.
        cnt += node->child[ith bit of N]->cnt.
        Post counting we will try to XOR the bit = ith bit of K
    if (ith bit of K == 0) then there is no option but to make the XOR 0, else
        we will exceed the value of K.
        node = node->child[ith bit of N].
    node = node->child[ith bit of N].
```

(GOOGLE) [Maximum Strictly Increasing Cells in a Matrix]
(Amazing trick) (LEETCODE)

d) Given a n*m integer matrix. You can select any cell in the matrix as your starting cell.

From the starting cell you can move to any other cell in the same row or same col, but only if value of the new cell is strictly greater than value in the current cell. Return how many maximum cells you can visit in this way.

Example)



(ans 4).

Soln

[Step1] → Create a map

map<int, vector<pair<int,int>>>.

this is going to be used for storing value → positions list. [Since there can be duplicate values]

[Step2] → We are going to start travelling the values in in creasing order.

We are going to keep 2 arrays :-

1) Row_max[i] → to store what is the maximum value I can get from ith row.
2) Col_max[j] → to store what is the maximum value I can get from jth col.

Code →

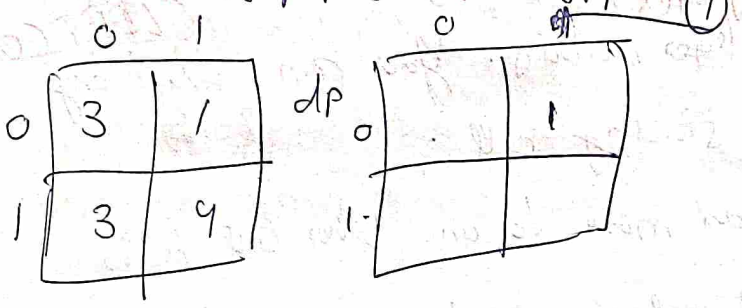
```
for (auto it=mp.begin; it!=mp.end(); it++){
    for (auto p : it->second){
        int i = p.first
        int j = p.second
        dp[i][j] = max(row[i], col[j]) + 1;
    }
}
```

Explanation of why updating rowmax and colmax in a diff loop in next page

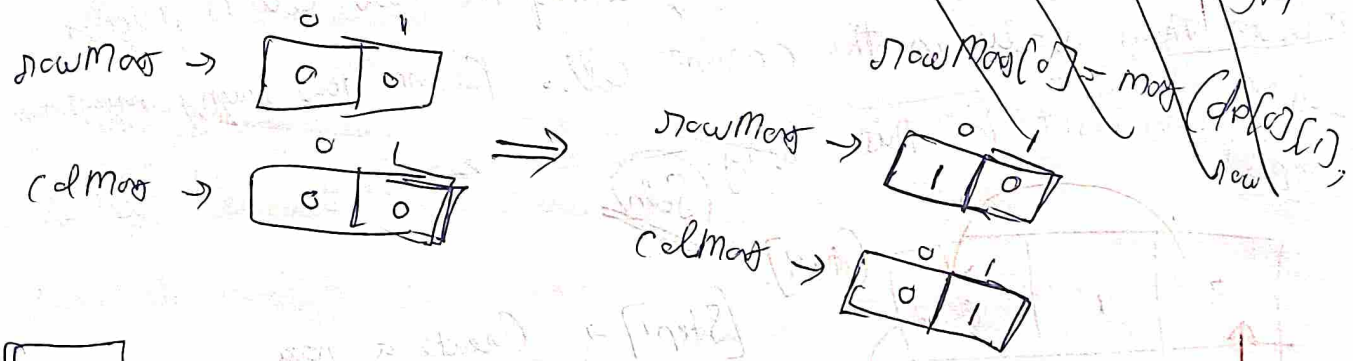
(V. V. Imp trick to handle duplicates)

```
for (auto p : it->second){ int i = p.first; int j = p.second;
    row_max[i] = max(row_max[i], dp[i][j]);
    col_max[j] = max(col_max[j], dp[i][j]);
}
}
```

Lets take the example (Reason) Step ① we start with



1 → (0,1).



Step 2

Next up we update 3. [Duplicate]

Now lets say we update rowMax and colMax after the first 3 is processed it self.

what happens then?

$dp[0][0] = max(rowMax[0], colMax[0]) + 1 = 2$

$rowMax[0] = max(rowMax[0], dp[0][0]) = 2$

$colMax[0] = max(colMax[0], dp[0][0]) = 2$ ⟹ {keep this part in mind}

Now when we are processing the next 3.

$dp[0][0] = max(rowMax[1], colMax[0]) + 1 = 3$ → which is wrong.

Because we updated the rowMax and colMax using the first 3. The 2nd (3) is now thinking there is a number smaller than itself on the row & col who can visiting max(rowMax, colMax) cells.

Hence.

First we should find answers of all the dfs(3). and then update row_Map & col_map.

Hence the 2nd loop.

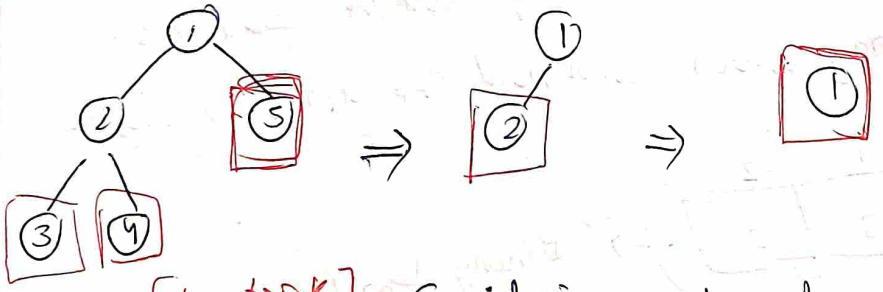
~~(LFU Cache)~~

~~(1) Lets all know what a LFU Cache is. If you have forgotten it.~~

(Find leaves of Binary Tree)

Create an unordered map (int → vector<int>) → to store level to leaf node map

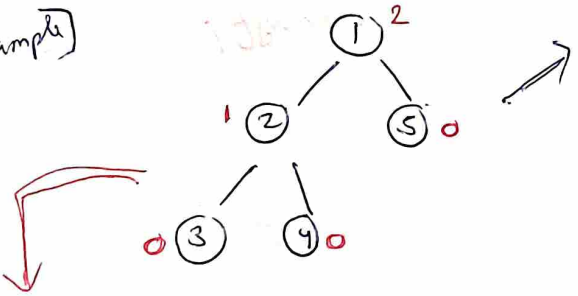
(1) You are given a binary tree, you need to remove ~~leaves at~~ each stage. A new tree will be formed and we need to remove its leaves again. At each step ~~we~~ store the removed leaves



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

Sol → [New trick] Considering each node as the root of its subtree we try to calculate its height. The nodes with the same height will be removed at the same iteration.

[Example]



vector<vector<int>> res;

res[level].push_back(node->val);

Code →

```
int dfs(root){
    if(root==NULL){return -1;}
    int level = max(dfs(root->left),dfs(root->right)) + 1
    res[level].push_back(root->val);
}
```

(Medium) (RLE Iterator) (Binary Search Approach)

O(logN)

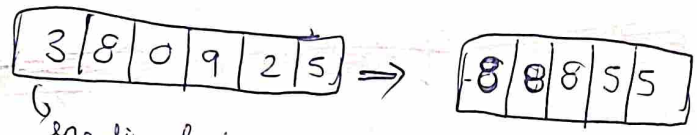
You will be given an encoded array where

i%2 == 0 → frequency of the number

and (i+1) → is the actual number.

→ there is a next function.

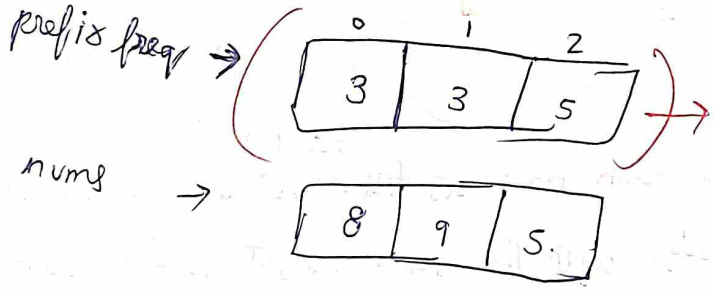
which tells you to extract next n numbers from ~~the~~ array and ~~print the~~ print the last number extracted.



encoding looks something like

Soln

We will be using binary search to solve this problem, we will be using a prefix sum of all the frequencies and created another array which will hold the actual numbers



Binary search for (freq[i] ≥ cnt) for lowest i

~~int~~

cnt = 0

int next(int n){

check if (cnt + n > sum of all elements) {return -1};

else → Binary search on the prefix freq for (cnt+n) and find the smallest index with that value.

(V.V.V. Hard) # Expression Add Operators [LeetCode]

You are given a string num, that only contains digits, and an integer target. You need to return all possibilities to insert binary operators '+', '-' and/or '*' between the digits of num so that the resultant expression evaluates to target value.

Example: [num="123", target=6]
output → ["1*2*3", "1+2+3"]

Note the operands cannot have leading zeroes.
operand can be "0" but not "00"

[Code In LeetCode] [Beware if you see this in interview]

Soln

This is going to be a crazy solution. Calm your mind before understanding it.

Step 1 Here we will be using the substring generation from [i----j]th subarray

for every recursive iteration we will have a start index.

```
f(startindex){
  int num = 0
  for(i = startindex; i < s.length(); i++){
    if(i - startindex > 1 and s[startindex] == '0') break;
    num += 10*num + s[i] - '0';
    string exp = s[start_index ... i]
  f(i+1);
}
```

→ we don't want leading zeroes.

⇒ this is how we will be generating subsegments

(Step2) Consider there were only '+' and '-' operator so no problem of using bodmass or precedence operator.

In that case ⇒ V.V.Imp

→ Very Simple if only + and − are there but if "*" is also there then we need to store another variable called prevNum.

```
void f(int i, string path, long result){
    if (i == originalString.length()){
        if (result == target) { path is one of the answer; }
        return;
    }
    curString = "";
    num = 0
    for(j=i; j < s.length(); j++){
        curString += s[j];
        num = (num*10) + s[j] - '0';
        if (i == 0) {
            f(j+1, path + curString, result + num);
        }
        f(j+1, path + "-" + curString, result - num);
    }
}
```

⇒ Choose to add.

⇒ Choose to subtract

Suppose till Now the expression way. 1 −2 and to be evaluated expression is 1−2*3. So prevNum needs to be stored.

prevNum = −2.

res = 1 − 2.

⇓

new res = res − prevNum + prevNum * (3)

Step3 Here we will modify the function to incorporate the multiplication changes

```
void f(int start, string &s, string path, long result,
       long prev, int target){
    if (start == s.length()){
        if (result == target){
            ans.push_back(path);
        }
        return;
    }
    string cur = "";
    long num = 0;
    for(i = start; i < s.length(); i++){
        cur += s[i];
        num = num*10 + s[i] - '0';
        f(i+1, s, path + "+" + cur, result+num, num, target);
        f(i+1, s, path + "-" + cur, result-num, -num, target);
        f(i+1, s, path + "*" + cur, (result - prevNum + prevNum*num), (prevNum*num), target);
    }
}
```

(result - prevNum + prevNum*num) ⇓ (V.Imp)

-num ⇓ V.Imp

prevNum*num ⇑ V.Imp

add Case

subtract Case

multiply Case

For the case

(1 + 2*3) → 1|2|3

prevNum = 2

num = 3

result = 1+2

For the case

(1 - 2*3)

prevNum = -2

num = 3

result = 1 - 2

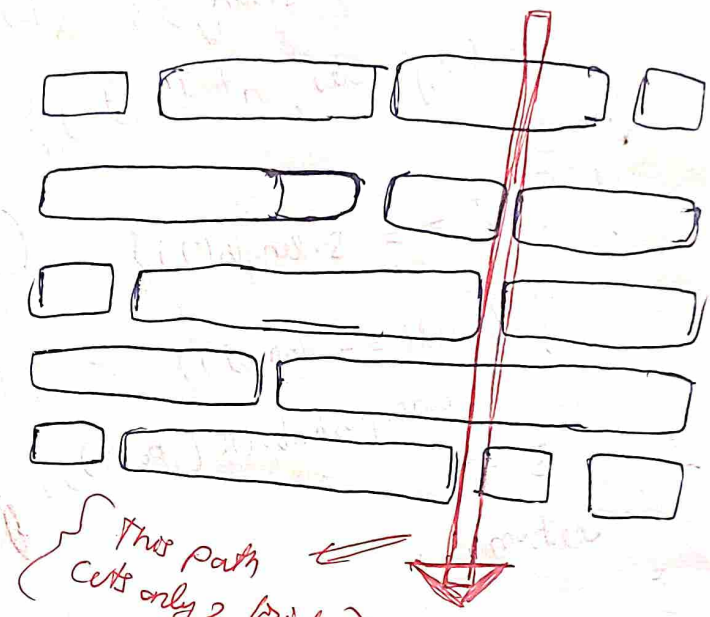
Brick Wall

(GOOGLE) (EDGE MARKING APPROACH) (MEDIUM)

Q There are bricks filled in front of you, for n rows. The i^{th} row has bricks of width 1.

Ex.

[1 2 2 1]
[3 1 2]
[1 3 2]
[2 4]
[3 1 2]
[1 3 1 1]



We want to draw a line across the brick wall from top to bottom is such a way that is cut the least number of bricks.

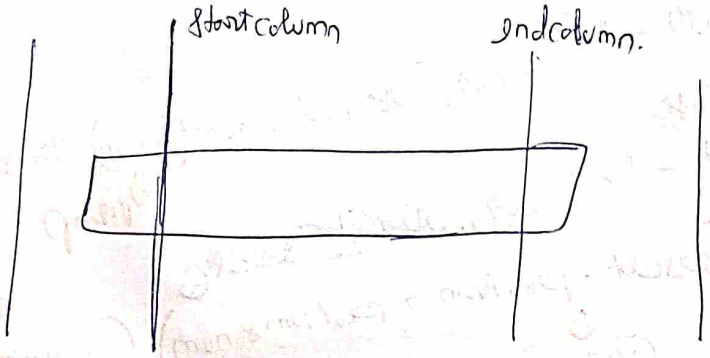
Soln There are 2 solutions for this.

[Soln 1] {Line Sweep}

After for each row using prefix sum, we calculate which column has lowest freq.

```
line[startcolumn] ++
line[endcolumn+1] --
```

* We will travel through a row and for every brick we will calculate its starting column and ending column the brick is covering.



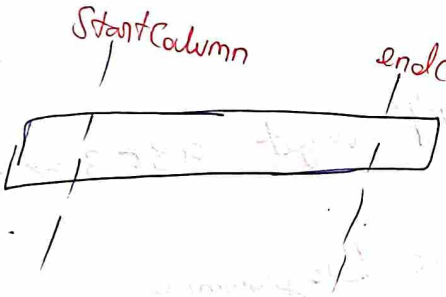
Soln 2

Now the line sweep technique has a problem.

Lets say the wall[i][j] $\leq 10^9$ so you cannot ~~~~ Store the startcolumn or endcolumn in the array

Approach 2

we will use a hashmap to mark the freqency of edges which are not touched.



freq[startColumn -1] ++

freq[endColumn +1] ++.

Now in the after we find what max(freq[i]) we can get.

maxEdges = max(freq[i]).

No of bricks = walls.size() = n.

Ans = (No of bricks - maxEdges)

Notes Trick

Minimum Difference between Largest and Smallest Value in 3 Moves.

Q) Given an integer array, you can modify atmost K numbers to any numbers of your choice. Return the minimum difference you can get after applying the operations between largest and smallest value.

Ans (Possible options) → First sort the array

1) discard K smallest numbers → Find (A[K] - A[n-1])
2) discard (K-1) smallest 1 largest Find (A[K-1] - A[n-2])
3) discard (K-2) smallest + 2 largest Find (A[K-2] - A[n-3])

Choose minimum of these.

(HARD) (String Compression II) [CODE IN LEETCODE]

Q) RLE (Run Length Encoding) (GOOGLE) is a string compression. Given a string s = "aaabcccd", k = 2. Notice how for single characters we don't append 1. Given an integer k, you can remove atmost k characters from the string s. Return the minimum length of RLE encoded string you can form.

RLE = a3bc3d

A) s = "aaabccd" k = 2 remove 'b' and 'd' we get a3c3 ⇒ 4.

* We are going to solve it using dynamic Programming

*) At each character we have a choice whether we want to take this character in our string or not.

*) We will keep calculating the length of the RLE at each index dynamically

State of dp.

dp [start][prevchar][len][k]

What is the minimum length of RLE we can form given that

① for the substring s[start, ... n-1], with the ② prevchar as the previous character of the string and ③ len as the length of previous continuous segments and ④ k = atmost elements we can still delete.

f(start, prevchar, len, K){

if(K<0){return INT_MAX;} → we deleted more than K characters

if(start >= n){return 0;} ⇒ we got to the end correctly

if(n-start ≤ K){return 0;} ⇒ Another optimisation if no of characters left < K. Then we delete all the characters.

if(dp(currentstate) != -1){return dp(currentstate);}

We decide to delete s[start] and not keep it.

choice1 = f(start+1, prevchar, len, K-1);

prevchar remains same

len of previous segment remains same

(1 character has been deleted so K→K-1)

We keep the character.

But there are 2 parts.

Part1 → if(s[start] == prevchar)

same character to the segment has been added.

choice2 = isLengthIncreasing(len+1) + f(start+1, prevChar, len+1, K);

{this is to check aa to a1b transition}

Part2 → if(s[start] != prevchar)

choice2 = 1 + f(start+1, s[start], 1, K);

length of segment = 1

prevchar now becomes s[start]

this 1 is added because a new character has been introduced after a long subsegment

aaab ⇒ a3b (3→+1)

[HARD] (★★★★) Finding MK Average [GOOGLE]

O(1) → calculateMK()

O(logm) → To add a number.

You are given two integers m and K and a stream of numbers is flowing in. We need to find MK average for this stream.

Steps in which MK Average is being calculated.

→ If the current no of elements in the stream is less than m then we return -1.

→ if no of elements we have encountered till now is greater than m., ① then take the latest m elements → {this is the part that we want to check} {keep in mind latest m elements}

② Remove the smallest K elements

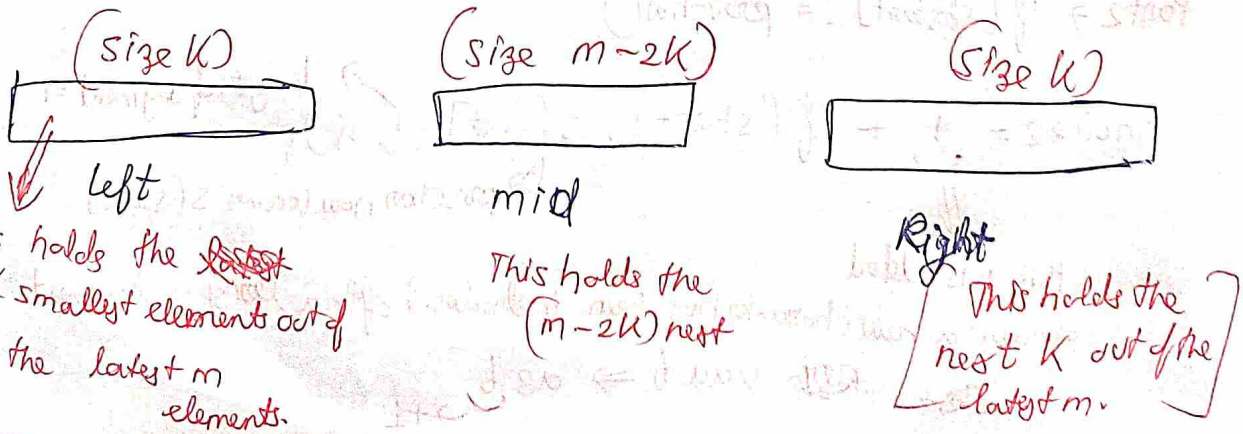
③ Remove the largest K element.

④ Calculate average of the remaining elements

[Soln] We will be using 4 variables.

① Queue → [ ] q [this will contain the latest m items]

② 3 multisets.



Add Number

if (q size is less than m){

push it to the queue.

Push this new number into the 3 multisets.

① push new number into left set

② check if Leftset size is exceeding in size more than K. if you want to delete the last element of the multiset

If Yes. st.erase (prev(st.end())) → this function will point to iterator previous to st.end()

then take the last element of leftset and push it into the mid set. ***

st.rbegin() ⇒ this iterator will take us to the end of the multiset

check if the mid set size is exceeding in size more than (m − 2K). ⇒ [update the sum of mid set accordingly]

~~check if Right set size is~~

If yes

then take the last element of mid set and push it into the Right set.

§.

Now the condition is that (q.size == m)

then ~~take delete~~

- q.front() has to be popped. = {delete_element}

and q.push (new) element

{P.T.O}

Now the {delete_element} needs to be deleted from the {left, mid or right} set where we want this element to be removed from. So that it is not in one of the latest n elements.

[n = delete_element]

{check if [*left.rbegin() >= n] erase left.erase(n)

else if (*mid.rbegin() >= n) erase mid.erase(n)

[sum -= n]

else right.erase(n)

[After erasing from one of the sets we need to do resizing again.

if [left.size() < K] left.insert(*mid.begin()).

mid.erase(mid.begin()).

sum -= *mid.begin()

if [mid.size() < m-2K]

mid.insert(*right.begin())

right.erase(right.begin()).

sum += *right.begin()

[We keep modifying sum on the fly]

(Amazing trick)

HARD

Optimize Water Distribution in a Village

[Imp to check why my approach failed]

GRAPH

There are n houses. For each house i, we can either build a well inside the house directly or connect the house to a component which has atleast one house where a well is built. You are given an array wells, where wells[i-1] gives us the cost to build a well at node i. Also you are given pipes edges, where its has [house1, house2, cost]. Basically cost to build a bidirectional pipe from house1 to house2 has a cost → cost. Find the minimum cost to connect all the edges.

Here we want to use DSU to connect the components

Soln

Approach is that you travel the edges sorted by their edge weight. ~~We will travel the edges and add~~

[Minimum Spanning Tree] → [After connecting all houses to house(0) using wells[i] cost.]

Trick

Step 1

[You assume a house 0 which only has water. We consider new pipes to be built from house[i] to house(0). In the end we want to see if the nodes connected have 0 in their component or not]

while checking a pipe, check node u and node v.

Case 1

if both the nodes have no water, then we will assign well to the node which has lower well setup cost and connect the edge and mark the other node as HAS WATER

Case 2

if both the nodes have water no need to connect

Case 3

if one of the nodes has water, then we simply just connect the node which has no water.

P.T.O

My Approach

1) Sort the nodes by their well setup cost.

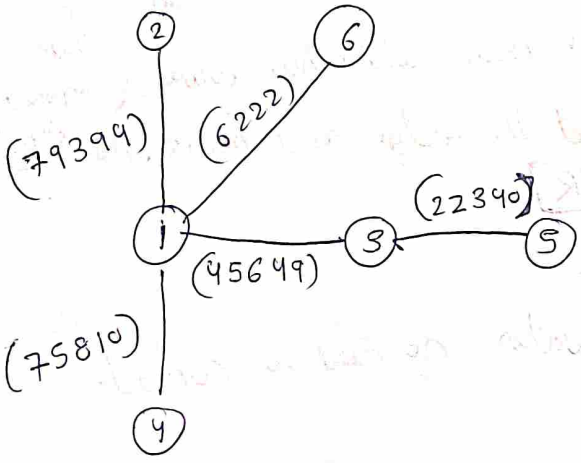
2) While ~~building the edge~~ travelling through the nodes, we want to check if node has water or not.

then using bfs as the source node as the ~~node~~ current node where we built the well. Then start merging edges if the cost to build a well on the new node to be stitched. ~~is~~ is greater than edge weight then we shall ~~not~~ plan to ~~stitch~~ stitch it.

[cost to stitch > cost to build well on this node]

Why is this approach wrong?

Test case 1:



well setup cost	
1 →	4625
2 →	65696
3 →	86292
4 →	68291
5 →	37147
6 →	7880

Here if you go by bfs then you will merge heavier weights first

(Atlassian)
(Hard) [Leetcode] [Maximum Path Quality of a Graph.]
Back Tracking
(Complexity Analysis) [Use of Constraints]

Q) You are given an undirected graph. From nodes 0 to n-1. You are given edges $[u_i, v_i, w_i]$ which tell w_i is the time required to go from $[u_i$ to $v_i]$ & $[v_i$ to $u_i]$. You are also given an array values[], where values[i] is the value of the i^{th} node. ma[illegible]

You need to tell what is the maximum score you can get while travelling from node 0 and ending at node 0 again within maxTime.
Score is sum of the value[i] of the distinct nodes visited.

Constraints →
$[10 \leq w_i, maxTime \leq 100]$
$n = 1000$
From each node 4 edges can be connected at max.

A) This problem is a game of constraints only.

What if we madly travel around the graph across all edges.
With 1 condition, we keep on adding the edge weights as we keep travelling around the graph.

Step 1 → Keep adding edge weights as we roam around the graph and stop traversing as soon as [time of traversal > maxTime]

Step 2 → keeping a visited array. Note we are not checking the visited array to see if we want to visit it or not. Rather we are using it as a counter array to keep track that how many times are we visiting it. If it already visited update its score, don't add

PTO

(Time Complexity Analysis)

→ Base Case → if (timeTravelled > maxTime) return.

* You see in constraints [10 ≤ wi ≤ 100]

So depth of each dfs can be of [level 10]. [100/10].

(i.e.) [maxTime/min(wi)] → worst case

and each node has max 4 edge connected to it. So at each depth level we have 4 choices.

So total we are doing 4 x 4 x . . . . . 4 (i.e.) (4^{10})

(10 steps operation)

*

And while traversal if you are seeing that the node is 0.

then update to maximum score

Similarly we can find the distance as well.

i.e.→ distance[node][i] → distance from node to its 2^{i} ancestor.

Now if we have this data. Then we can find distance between any two nodes. Node (a, b).

distance (a) to distance (LCA(a,b))

+

distance (b) to distance (LCA(a,b)).

Amazing concept

Using LCA to find max Edge Weight between two nodes.

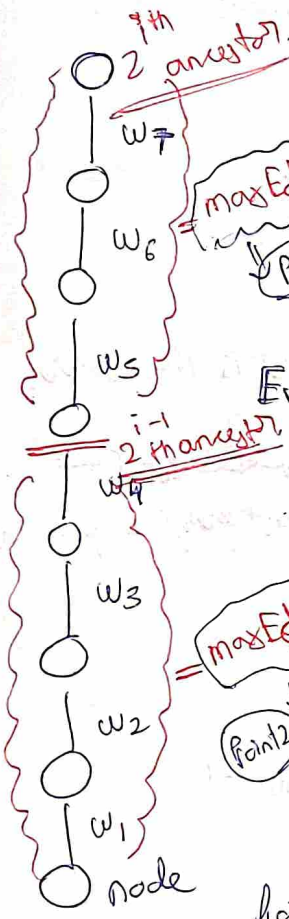
[Find any two thing [sum, xor, max] between two nodes.]

This is an amazing concept using LCA. and can be used to find amazing properties between two nodes.

{You can find within nodes as well. Suppose you are asked to solve xor of edgeweights between node A and node B}.

Such as distance,

Now we can store



maxEdgeWeight[node][i] => this tells me the maximum edgeweight encountered in the path from node to its 2^{i} th ancestor.

maxEdgeWeight[node][i] = maxEdgeWeight[up[node][i-1]][i-1]

Point1

Evaluation maxEdgeWeight[node][i] =

maxEdgeWeight[node][i-1]

Point2

```
dfs(node, parent, edgeweight){
    up[node][0] = parent
    maxEdgeWeight[node][0] = edgeweight;

    for(i=1; i<logi; i++){
        if(up[node][i-1]==-1){
            up[node][i] = -1
        }
        up[node][i] = up[up[node][i-1]][i-1]
        maxEdgeWeight[node][i] = max(maxEdgeWeight[node][i-1],
                                     maxEdgeWeight[up[node][i-1]][i-1]);
    }
```

Largest Color Value in Directed Graph

Imp

(Revised)

(Thought Process of Directed Graphs)

Q) There is a directed graph of n nodes and m edges. nodes from 0 to n-1. For each node you are given colors. Directed edges are also given.

S = "abcde" which means-

0	1	2	3	4
a	b	c	d	e

color[0] = a
color[1] = b
color[2] = c
color[3] = d
color[4] = e

A path from node U to node V is a path if U to V is reachable.

Also for each path there is a score.

$[U \to v_1 \to v_2 \to v_3 \to \dots \to V]$

which ever colors frequency is maximum, that is the score.

In the path from node U to node V, each node has a colour. If you count the freq of each color then the highest frequency is the answer.

Note: Return -1 if there is a cycle in the graph

(Step 1) → Detect a cycle. If cycle is there then return -1

A)

(Step 2) Trick is Topological Sort.

Whenever it's a directed graph try to check if it's solvable by topological sort or not

But Now in BFS queue of topsort store an array of Length 26, which has the frequencies stored.

At each node check what is the max of that array

Few Categories of questions,

→ Upon seeing these type of questions what is the thought process that should come in your mind.

[Categories I] [Swap to Sort]

Q) Given an array you will be given a condition on which certain indexes and certain values are swapable amongst themselves. You want to find the lexicographically smallest array that you can find using these swaps.

A) The first thought that should come to your brain is how you can group the indices which are swappable amongst themselves ~~Once you have~~ [Note:] Don't try to manually sort the values.

Once you have grouped the set of indices into 1 value using DSU. Sort only those elements and insert into their respective position.

(Category II) [Interval swap and Sort]

Q) (a) You will be given intervals [start, end] and you will be asked to how many minimum meetings room will be required.

A) Either used priority-queue or line sweep to solve this question. But to solve this using line sweep we will be needing all the starts and end, [end-start] to be in an array formable range.

[Category III] Given a graph and path between two nodes you might be asked to find sum of nodes or max of something within that path.

(A) Think of LCA and binary Lifting during that. Find ways to calculate things from node(a) to LCA(a,b) and from node(b) to LCA(a,b).

[Longest Consecutive Sequence] (V. V. Imp) (Trick) (REVISE) (GOOGLE)

{You need to do this question in O(N)}

(Q) What is the longest consecutive sequence of an array

arr → [100, 4, 200, 1, 3, 2]

9 1 3 2

are the list of consecutive elements.

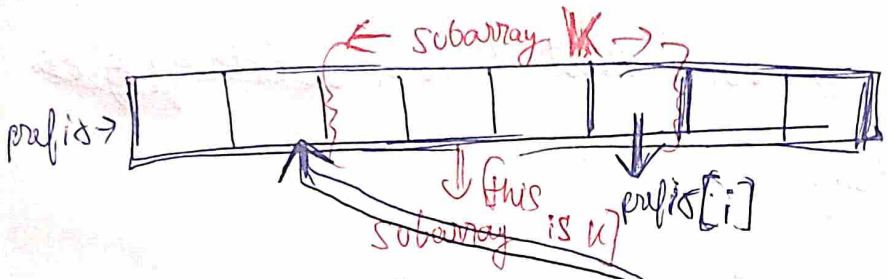
(A) Lets walk through the a naive approach. Keep a map or hashset of all the numbers present in the array. For each nums[i], keep doing a +1 and check if (nums[i]+1) is present or not then check (nums[i]+2) is present or not. If not then break, else keep increasing the length of the segment

Optimal → The above approach which came to my mind was $O(N^2)$. Now there is a slight fix. For every nums[i] just check if (nums[i]-1) is present in the hashset or not. If present then don't evaluate it. If not present, then start evaluating to check if nums[i]+1 is present or not then (nums[i]+2) and so on. This way we are only evaluating the start of the sequence.

[Category III] → this is a way to

Subarray having sum = k

Create a prefix array: Lets say $\{-10^9 \le a[i] \le 10^9\}$



In the map keep storing prefix sums till [i-1].

Find if there was a occurrence of [prefix[i] − K]

[Things to keep in mind while coding]

If you have a set which has integers in sorted ~~order~~ order.

You want to do an operation of deleting a few numbers from the set.

You might do (while(it != st.end()){ st.erase(it) and it++;} → NULL Pointer exception)

make sure to move the pointer and then call erase.

```
while(it != st.end()){
    num = *it;
    it++;
    st.erase(num);
}
```

→ [Else iterator to the starting number will be lost].

[Minimum Window Substring]

GOOGLE

N·Imp

Q) Given a string S and T. Find the minimum contiguous substring ~~sequence~~ of S, such that T is a ~~substring of~~ subsequence of the substring.

i.e. T can be generated using the subsequence of the substring of S.

A) Idea behind the solution : [sliding window]

S → a b b d b b a e a d e

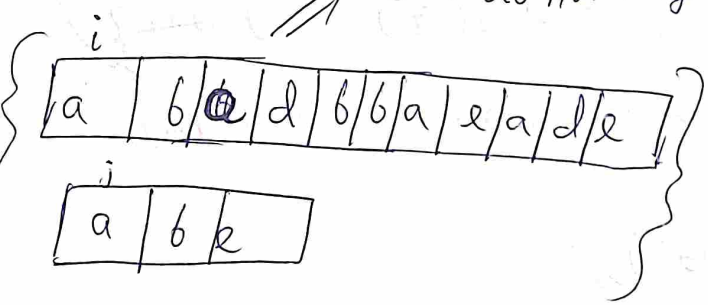
T → a b e

(Similar Questions)

keep two pointers i & j

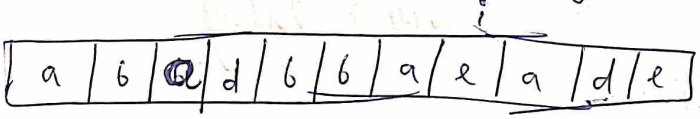
Step1

As we do normally

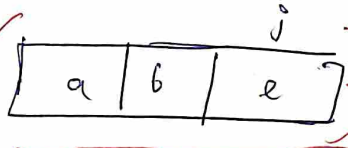


Step2

Now suppose at i we found that string T is a subsequence.

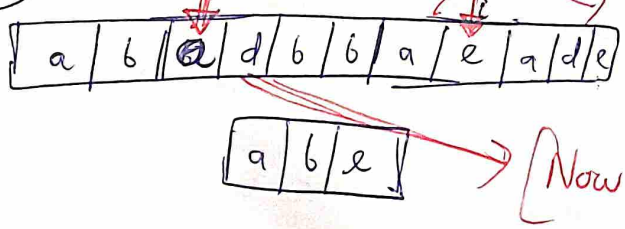


→ [And keep reducing i till we find first occurence of substring of i]



→ Now take j at the end of T.

Step3



→ Found occurrence of abe string

→ [Now shift the window to here and do the same process]

(***) (GOOGLE) Category Question. (Very Imp) (way of thinking these questions)

[Maximum Performance of a Team]

(QQ) You are given two arrays of size n. Speed and Efficiency. Speed[i] and Efficiency[i] represent speed and efficiency of the i^{th} Engineer. We can choose at most k Engineers such that the team's performance is maximum.

Team's Performance = (Sum of the speed of all Engineers) * minimum Efficiency of all those Engineers

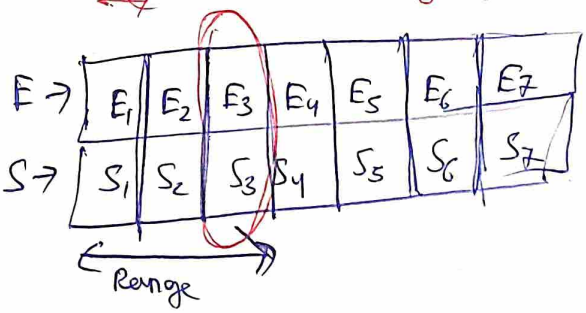
(A) → [Use MinHeap for this] → [Try to think of these questions in this manner]

How to do these kinds of Problems?

(Step 1) [Sort the Engineers by their decreasing Efficiencies]

Thought process → Try to think of it as if you are considering a particular Efficiency as minimum at the moment. (Sorted in descending order)

(Step 2) Lets say we are choosing atmost k Engineers such that minimum efficiency is E_3.



(Step 3) ~~Choose~~ Now we have to choose atmost k such speeds. within the given such Range and pop the minimum speed out of the current and insert the new speed.