3. Longest Substring Without Repeating Characters

in this problem we require data structures like map (mp) and 2 main variables prev and index i for traversing string initially map is empty and while traversing with index i there is condition if(mp.find(s[i])==mp.end()) i.e if string letter s[i] is not present in map then insert that letter as key and value as that index of letter else if you find the same letter in map then that means that letter is repeated at index i and that previous letter is present at index mp[s[i]] so by substracting current index with mapped index we can get length of index here we use another variable maxi which only store the maximum length which is found by above procedure i.e by maxi = max(maxi,len); then we clear the map and change i as idx index found in map and previous as index found in map+1;

and finally when for loop end we again compare size of map as map is storing till unique letters and maximum element till now.... and return maxi

---

5. Longest Palindromic Substring

in this problem they didnt think how to store all strings they just think on indices they change the indices accordingly and kept a end variable which stores maximum length of palindrome

here there are two possibilities one is of even palindrome and another is of odd palindrome

first we have to traverse the array from index 1 and consider each element while traversing is an middle element of palindrome then for even element if i is variable by which we are traversing then l =i-1 and h = i where l and h are variables which shows assumed starting and ending of palindrome so before traversing we have already assigned start and end index of palindrome as 0 and 1 respectively while traversing if we find elements like s[l] == s[h] then we change start as l and end as h-l+1 and then... decrement l and increment h

---

6. Zigzag Conversion

in this program you have to convert given string into zig zag pattern in given number number of rows

observe each column so there is one basic rule of zigzag travelling

11. Container With Most Water

In this problem we first assign three variables low , maxi and res as 0 and high = hight array size -1; then we use while loop which iterates till low< high we find minimum element between height[low] and height[high] and store in mini then the amount of water they will contain will be mini * (high-low) which will be stored in res variable then we use if statement like if(res>maxi){ maxi = res; } then we have take decision either we have to increase low or decrease high we increase low if height[low] less than height[high} or else decrease high and finally return maxi

15. 3Sum

we have to find three elements whose sum is zero

sort the array in increasing order declare three variables lo hi and a start traversing from begin and assign for loop element as a for a element assign lo = a+1; hi = nums.end-1; and sum as negative of a and traverse between lo and hi with condition as lo + hi ==sum then push lo , hi and a in vector

here basically we are decreasing one loop by assigning as we require sum as - *a and assign lo = a+1 and hi as nums.end()-1;

so if this lo + hi ==a then we have to push lo hi and a in ans vector

26. Remove Duplicates from Sorted Array

not underrstanded at all

class Solution { public: int removeDuplicates(vector& nums) { int length=nums.size(); int i,j=0; if (length==0) return 0; ` for(i=0; i<length; i++){ if(nums[j]!=nums[i]){ j++; nums[j]=nums[i]; } } for(i=0; i<length; i++){ cout<<nums[i]<<" "; } return j+1; } };

31. Next Permutation

in this problem we have to find next permution of given array formed number next permutation means if you given array 1,2,3 thenn we have to return number 2,1,3 which is next possible number from given numbers of array for that first traverse from end of the array and find the array's index who is greater than i+1th array number by using while loop if i is greater than or equal to 0 then again traverse from end till you find jth element who is less than ith element then swap element i and j

after finding i and j element reverse the array from (arr.begin()+i+1, arr.end()); and return the result

42. Trapping Rain Water

traverse the array from begining push the maximum element till now while traverising in prefix vector now again traverse the array from end now push the maximum element till now while traversing in suffix then reverse the suffix array and then traverse the height array while traversing find the min betwenn int n = min(prefix[i],suffix[i])-height[i];
add n to sum

45. Jump Game II

in this problem we first deside two varables l and r these are the range to which we can jump and another variable named farthest which shows maximum element to which we can jump and while jumping to maximum element choose farthest number by comparing farthest and i+nums[i];

47. Permutations II

vector<vector> v1(v.begin(),v.end());

49. Group Anagrams

in this algorithm we create frequency table of each charcter in string by using map and then also create bigmp which contains map as key and value as vector of string then after creating frequency table (map of characters) we try to find same map table in bigmap if big map dont contain then we insert that map with vector which contain string whose characters are mapped...if this type of is already present in big map then we insert that str in vector of bigmap having same map.

53. Maximum Subarray

in this question

54. Spiral Matrix

In this problem we take to array which gives only direction by incrementing and decrementing row or column we just add this directions to our x and y... we have to change direction i.e we have to increment dr and dc array when x and y goes out of matrix... or when this newx and newy are visited

55. Jump Game

in this problem we first create a variable named goal which is last index of array then we see with that index can we reach the goal (as while starting goal is last index) if we reaching with that element at last index (goal) or or greater than last index(goal)we set goal is that index and decrease index and check can we reach that new goal with this element + index

56. Merge Intervals

in this problem we have to create separate intervals means index 1 of any one interval should be greater than index 0 of the same interval and that same index element should be less than index 0 of other intervals.

so for that first assume 0th whole interval as the temp interval. use this tempinterval and traverse through all intervals one by one and see if index 1 of temp interval is greater than or equal to traversing intervals zeroth interval if it is then substitute temp intervals 1th index as max between temp intervals 1 index or traversing intervals 1st index...else push the whole tempinterval in final mergedinterval and replace tempinterval as current traversing interval. and finally after traversing whole interval return merged interval array....

76. Minimum Window Substring

in this problem we have to use map datastructure in which we store frequency of pattern is stored and the idea is we have to traverse using 2 while loop using j and i variable initialised with zero first while loop decrease the mp[s[j]]-- and if the mp[s[j]] == 0 then we decrease the count here count is length of pattern string then doing this in while loop once you find count becomes zero means we have all the characters which are required from index 0 to current traversing index j now we start our second while loop which start to decrease window by increasing i index this loop traverses till count ==0; here first if statement comes up with condition **if (len > j-i+1)** then save len as len = min(len , j-i+1); where len represent last index of ans string and start as index i which is starting index.

```
finally return return s.substr(start,len);
```

118. Pascal's Triangle

In this problem create vector of rows and columns = required no. of pascal triangle (numRows) and traverse through the the vector if the traversing index is 0 or col-1 then put their 1 or else put their

pascal[i][j] =pascal[i-1][j-1]+pascal[i-1][j];

```
 and again traverse through same matrix and remove zeroes one by one
```

121. Best Time to Buy and Sell Stock

we create two variables maximum profit and minsofar
in this problem we traverse the array from begining and think to sell the stok at the same traversing array element by subtracting minimum sofar element with current traversing element and store it in maximumprofit element `maxprofit = max(maxprofit,profit)`

125. Valid Palindrome

in this algorithm two pointers are used one is pointing to start and one is to end start is gets incremented till it goes to alnumeric character and end also gets decremented till to goes to alnumeric character and then it check start letter and end letter if they are not same then return false else again increment start and end and repeate above methods...

152. Maximum Product Subarray

declare 3 variables as ma ,mi,ans with value as nums.begin(); and traverse the array nums in this problem the main problem is to decide that should we include the -ve nums in sum or not as if there are two negative numbers then their product will be max otherwise if their are 1 or odd number of -ve then the product will be minimum hence algorithm is ... while traversing store the values in ma as ma = max(*i,ma * *i) and if there is negative number then values of ma and min are swapped then maximum values will stored by min by including negative number and when again negative comes swap the values of ma and mi

i.e. ma will be storing product of +ve numbers as soon as we encounter negative value change value of ma and mi such that its value get preserved (as mi storing values which are minimum so the value which is get multiplied by -ve number will minimum number)

209. Minimum Size Subarray Sum

in this problem we use sliding window techinque in which we initialize two pointers start amd end as 0; then initialize a while loop in which we increment end and add these elements into current sum and when current sum becomes not less than targeted element i.e. sum becomes equal or greater than target element then start incrementing start variable and decreasing current sum

219. Contains Duplicate II

first map the elements with their respective indices and then sort the arrays and if arr[i]==arr[i+1] then return true...

238. Product of Array Except Self

first create an array which have i th index as product of numbers till i-1 th index in nums array then the last element of array will be at its correct value then then go to second last element of array then you have to multiply the second index value with current index -1 value * product of values till less than current index-2 value.

304. Range Sum Query 2D - Immutable

in this problem basic idea is change the array by first adding sum of left side array in current index then adding upperside array in current index and then the sum of rectangle is by formula

int total = matrix[row2][col2]; int extra = (col1!=0? matrix[row2][col1-1]:0)+(row1 !=0?matrix[row1-1][col2]:0)-((row1!=0 && col1!=0)?matrix[row1-1][col1-1]:0); return total -extra;

424. Longest Repeating Character Replacement

in this problem we have to find longest string possible of same characters by having ability to change string character for given number of times

for that we first initialize 4 variables l ,cnt,maxlength=1, i =0; then we traverse through while loop having condition while(i<s.length()) {i} will be the character which will determine end of the maxlength substring {l} will be the character which will determine start of the maxlength substring {cnt} will determine available character change if it is greater than k (permitted no. of changes) then we cannot change further characters of remaining string

first we have a string and maximum no of changes parmitted number k we try all the variables which we can change and place at given string means if we have given string ABBAB then we will start placing A at every place of string where the character at that string is not equal to A then we will start placing B .....placing D till our cnt variable do not exceed k (permitted changes) the main idea is that we will travel through string using while loop and i will be outer while loop condition iterator and if while iterating we find that character is not equal to testing character (ch) then we increase cnt and then use while loop if cnt > k if it is true then we check next condition if s[l] != ch then we decrease cnt (which will help to exit from this while loop) and at any cost we have to increase l after that we store the maximum length by maxLength = max(maxLength,i-l+1)

and after that we return maxLength variable

507. Perfect Number

a perfect number is a number whose sum of divisors is equal to number itself

523. Continuous Subarray Sum

it is storing the number required on particular index to become divisible by k

566. Reshape the Matrix

basic idea is 1st convert whole matrix to 1D matrix by mat3.push_back(mat[i][j]); where mat3 is simple 1d matrix. then convert it into required matrix by new_matrix[i / c][i % c] = mat3[i];

572. Subtree of Another Tree

In this problem we have to do 2 step

1) FIND the subroot->value in root (ie find root value of subtree in tree) 2)after finding check they are identical or not.

1) to find subroot in root we use below function if(areIdentical(root,subRoot)){ return true; } return isSubtree(root->left, subRoot) || isSubtree(root->right, subRoot); if root value and subroot value are not same then we call root ->left with subroot and root->right with subroot and then again call areIdentical then if it return true then they are identical

647. Palindromic Substrings

same as longest palindromic substring just initialize count instead of start and end index

724. Find Pivot Index

in this problem first calculate sum of all array element and store it in right_sum variable then start traversing in array do two things while traversing one is decrease the traversing element from right_sum variable and increase the left_sum variable by adding traversing element and while traversing if you find left_sum == right_sum return the (i) index of traversing

2383. Minimum Hours of Training to Win a Competition

So the problem is we have given initialEnergy and initialExperience and two arrays of component with in one there is energy and in another there is experience if you want to defeate that person then your energy and experience should be greater than that person

so if they are not then you will not win the competation if you want to increase your energy or experience by one then you can train yourself by one hour and if 2 then train by 2 hours so you have to output no of hours of training you needed

in case of energy we can easily calculate how much energy do we require to win competition it can be calculated by totalt = (energys - initialEnergy) +1; energys is total energys of all opponents

then there is problem about experence the experience will increase as we defeat opponent so it is not simple as energies so, we introduce new variable nex which represent current experence

if i ==0 and nex<= experence[0] then we can increase totalt by totalt = totalt + (experience[i] - nex)+1 ;

```
and current experience by

            nex = nex + (experience[i] - nex) + 1;

                                and one another if else
                                    if( i+1 < experience.size() && nex > experience[i])
        {
           nex = nex + experience[i];
        }


                    and also another loop

                    if( (i+1 < experience.size() && nex <= experience[i+1]) )
        {
         totalt = totalt + (experience[i+1] - nex)+1 ;
            nex = nex + (experience[i+1] - nex) + 1;

        }

    }
    if(totalt<0){
        return 0;
    }
```

2432. The Employee That Worked on the Longest Task

here problem is we have given employee id and there work time if the work time is same then return the employee id which is smaller so to find the work time of each emplyee we have to find difference between end of the work of previous employee and the current employee which is stored in **max** variable if this max is

greater than smax which is initially zero then replace smax as max and as max is just greater than smax place mp[i] as 2 and if it is equal to smax then place mp[i] as 1. after doing all these task create a maxi variable assign value INT_MAX to it and start for loop from i = 0 to logs.size() and if (mp[i]==1 && maxi > logs[i][0]) || mp[i] == 2 then maxi = logs[i][0]; and return maxi

2537. Count the Number of Good Subarrays

in this problem we first create a map which calculate frequency of each number in array when the frequency of ith number becomes greater than needed (i.e k) then we go inside while loop which has condition of whlle (i>=left and temp>= k) { ...rest of code }here temp is frequency of i th element

then in while loop we have another condition if (mp[nums[left]] > 1) then we decrease temp -= (mp[nums[left]] - 1); this is because as they are greater than 1 they are formed pair now we are decreasing that pair by increasing left and decreasing mp[nums[left]]--; so that we can chek does that decreasing pair causes overall subarray to remain pairs less than k or not