HashMap

~ By Kapil Yadav

LinkedIn

Problem 1 . Frequencies of Limited Range Array Elements

Given an array A[] of N positive integers which can contain integers from 1 to P where elements can be repeated or can be absent from the array. Your task is to count the frequency of all elements from 1 to N.

Note: The elements greater than N in the array can be ignored for counting.

arr = {1, 1, 2, 2, 3, 3, 4, 9}
N=8
P=9
Output : {2,2,2,1,0,0,0,0}

Brute Force : TC = $O(N^2)$ and SC = $O(N)$

1. Take an extra array arrb of size N, initialize all the elements with 0
   arrb = {0,0,0,0,0}

NOTE : 1.we will run I loop till i<N, not N-1, because it may happen that last element will be distinct.
2. we will skip the the pass for duplicate elements. (using continue)

Since array elements are starting from 1, but array is 0 indexed, so We will use k = arr[i]-1 to store the count of arr[i] in arrb.

arrb[k]=arr[[arr[i]-1]]

Example - arr = {1, 1, 2, 2, 3, 3, 4, 9}

i=0

For i=0, arrb[0] =0, means we have not counted for arr[i].

arrb[0]=1;

J=1,[1==1] , so arrb[0]++.
J=2, [1!=2]
J=3, [1!=2]
J=4, [1!=3]
J=5, [1!=3]
J=6, [1!=4]
J=7, [1!=9]
arrb[]= {2,0,0,0,0,0,0,0}

i=1

For i=1, arr[i]=1, we will check in our additional array arrb, , since we have already computed for value 1, we will skip it using continue.

i=2

Since arr[2]=2, and arr[k] is 0, which is not computed , so

arrb[1] = 1;
J=3, [2==2] , so arrb[1]++
J=4, [2!=3]
J=5, [2!=3]
J=6, [2!=4]
J=7, [2!=9]
arrb[]= {2,2,0,0,0,0,0,0}

And so on..
After all the passes, we will get output as:
arrb[]= {2,2,2,1,0,0,0,0}

Since we are running two loops one inside another, Time complexity of the program will be $O(N^2)$ and
We are using extra space to store N elements space complexity will be $O(N)$.

Code: for above approach :

```cpp
#include <iostream>
using namespace std;


void countfrequencies(int arr[], int n, int p)
{
    int arrb[p];
  for (int i = 0; i < n; i++) {
        arrb[i] = 0;
  }


for(int i = 0;i<n;i++)
/* i will run for i<n,  because if
last element is distinct it can cover it.
*/
{
      int k = arr[i]-1; //because output array is 1 index;

    if(arrb[k]!=0)
        continue;

    arrb[k]=1;
    for(int j=i+1;j<n;j++)
    {
        if(arr[i]==arr[j])
          {
                arrb[k]++;
          }

    }
}
for(int i=0;i<n;i++)
cout<<arrb[i]<<endl;

}



int main() {
    int arr[]={1,1,1,1,2,3,3,3,3,3,3,4,5};
    int size= sizeof(arr)/sizeof(int);
   int p=9;
   countfrequencies(arr,size,p);
}
```

Approach 2 : we can use hashmap (TC - O(N), SC - O(N))

arr = {1, 1, 2, 2, 3, 3, 4, 9}
N=8
P=9
Output : {2,2,2,1,0,0,0,0}

Element	Count
1	X 2
2	X 2
3	X 2
4	1
9	1

First iterate over the array and increase the count of elements in map, (unordered_map in C++),
Then run a loop till N, and check if element is present in the map or not,
If present, then store back the count in main array, else store 0.

Code: for above approach :

```
void frequencyCount(vector<int>& arr,int N, int P)
 {
 unordered_map<int,int> map;

     for(int i=0;i<N;i++)
         {
             map[arr[i]]++;
         }

  for(int j=1;j<=N;j++)
   {
       if(map.find(j)!=map.end())
          arr[j-1]=map[j];
      else
         arr[j-1]=0;
   }

 }
```

Problem-2, Hashmap

31 December 2021 21:00

Problem - 2 Check If Array Pairs Are Divisible by k

LinkedIn

Problem Statement :

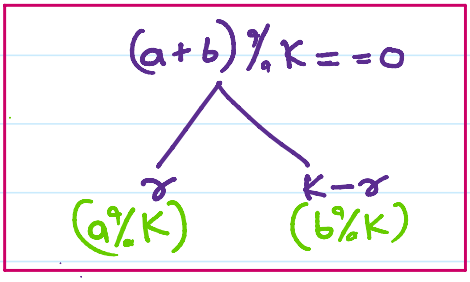
Given an array of integers arr of even length n and an integer k.
We want to divide the array into exactly n / 2 pairs such that the sum of each pair is divisible by k.
Return true If you can find a way to do that or false otherwise.

Example 1 :

Input:	arr = [1,2,3,4,5,10,6,7,8,9], k = 5
Output:	true

Solution : Firstly we will store the remainder (arr[i]%k) in the map,

1%5 = 1
2%5 = 2
3%5 = 3
4%5 = 4
5%5 = 0
10%5 = 0
6%5 = 1
7%5 = 2
8%5 = 3
9%5 = 4



We can observe that, to divisible by k, if one number is giving Remainder r, to make a pair we need a number having remainder k - r.

For example : *1%5 =1, so we will check the count of k -1 i.e. 4, If they are equal, they will make pair, check further.*

Now we will iterate the map, and compare the count of each key with (k - key),

- *If k - key, is not present in the map, means it will not form a pair, return false.*
- *If k - key is present in the map, check if the count is equal or not equal, if not equal return false;*
- *Here one edge case is, if remainder is 0, check if it is even or odd, if it is odd, then return false.*
  For (5,10) pair, 5%k and 10% k will give remainder as 0, so count should be even to make a pair.

Key (Remainder)	Value (Count)
0	2
1	2
2	2
3	2
4	2

Example 2 :

Input:	**arr = [2,3,5,10,6,7,8,9,12,14] k = 5**
Output:	false

Solution : Firstly we will store the remainder (arr[i]%k) in the map,

2%5 = 2	3%5 = 3	5%5 = 0	10%5 = 0	6%5 = 1	7%5 = 2
8%5 = 3	9%5 = 4	12%5 = 2	14%5 = 4		

Key (Remainder)	Value (Count)
0	2
1	1
2	3
3	2
4	2

For value 2, r =2, k - r = 3, count of 2 is 3, and count of 3 is 2.

return false;

NOTE : if the array length is odd return false, because odd numbers cannot make pairs.
arr = {5,15,20,25} , k = 10, map[5] = 3.
In this approach, we can observe that r=5, k - r is also 5 and every time, their count will be equal., so
how to handle this case? :
Well, we don't need to do anything, because 5 will try to make pair with 15 and 25, but 20 will remain unpair, 20%10 = 2 ,10-2=8, 8 is not present in map, so it will return false.
Still if want to handle this case, check for if k is even, check if map[k/2] %2 is even or odd, if it is odd, return false.

Code: for above approach :

```cpp
bool canArrange(vector<int>& arr, int k) {

        unordered_map<int,int> map;
        int n = arr.size();

        for(int i =0;i<n;i++)
        {
            int currentvalue= arr[i];
            int currentremainder = ((currentvalue%k)+ k) % k;
          //for negative values us mod like above.
            map[currentremainder]++;
        }


 for(auto it=map.begin();it!=map.end();it++){
            if(it->first==0)
            {
                if(map[it->first]%2!=0)
                    return false;
            }
                else if(map.find(k-it->first)==map.end())
                    return false;
                else if (map[it->first]!=map[k-it->first])
                    return false;
            }
         return true;
    }
```

Points : We can run loop k/2 times, but it will check everytime (k/2 times) whether key exists in map or not. For example arr = {98,2} and k=100, if map size will be 2, but loop will run 50 (k/2) times to check, so inefficient.

Problem - 3 Hashmap

31 December 2021 21:00

LinkedIn

Problem 3 : Largest subarray with 0 sum

Input:	N = 8 A[] = {15,-2,2,-8,1,7,10,23}
Output:	5
Explanation:	The largest subarray with sum 0 will be -2 2 -8 1 7.

Problem Statement : we have to find out the largest subarray having 0 sum.
{-2,2} , sum = 0, length =2
{-8,1,7}, sum = 0, length =3
{-2,2,8,1,7}, sum = 0, length =5, so output will be 5.

Brute Force I: $TC = O(N^3)$ and $SC = O(1)$

A[] = {15,-2,2,-8,1,7,10,23}

i = 0

j=0; k=0	{15}
j=1, k=0 to k=1	{15, -2}
J=2, k=0 to k=2	{15,-2,2}
J=3, k=0 to k=3	{15,-2,2,-8}
J=4, k=0 to k=4	{15,-2,2,-8,1}
J=5, k=0 to k=5	{15,-2,2,-8,1,7}
J=6, k=0 to k=6	{15,-2,2,-8,1,7,10}
J=7, k=0 to k=7	{15,-2,2,-8,1,7,10,23}

i = 1

j=1, k=0 to k=1	{ -2}
J=2, k=0 to k=2	{-2,2}
J=3, k=0 to k=3	{-2,2,-8}
J=4, k=0 to k=4	{-2,2,-8,1}
J=5, k=0 to k=5	{-2,2,-8,1,7}
J=6, k=0 to k=6	{-2,2,-8,1,7,10}
J=7, k=0 to k=7	{-2,2,-8,1,7,10,23}

i = 2

J=2, k=0 to k=2	{2}
J=3, k=0 to k=3	{2,-8}
J=4, k=0 to k=4	{2,-8,1}
J=5, k=0 to k=5	{2,-8,1,7}
J=6, k=0 to k=6	{,2,-8,1,7,10}
J=7, k=0 to k=7	{-,2,-8,1,7,10,23}

i = 3

J=3, k=0 to k=3	{-8}
J=4, k=0 to k=4	{-8,1}
J=5, k=0 to k=5	{-8,1,7}
J=6, k=0 to k=6	{,-8,1,7,10}
J=7, k=0 to k=7	{-8,1,7,10,23}

i = 4

J=4, k=0 to k=4	{,1}
J=5, k=0 to k=5	{,1,7}
J=6, k=0 to k=6	{,1,7,10}
J=7, k=0 to k=7	{1,7,10,23}

i = 5

J=5, k=0 to k=5	{,7}
J=6, k=0 to k=6	{,7,10}
J=7, k=0 to k=7	{7,10,23}

i = 6

J=6, k=0 to k=6	{,10}
J=7, k=0 to k=7	{10,23}

i = 7

J=7, k=0 to k=7	{10,23}

Code: for above approach :

```
int maxLen(vector<int>&A, int n)
   {    int maxlen = 0;

   for(int i = 0; i<n; i++)
   {
      for(int j = i; j<n;j++)
      {
           int sum = 0;
           for(int k = i; k<=j; k++)
               sum = sum + A[k];
           if(sum == 0)
           {

               maxlen = max(maxlen,j-i+1);
           }
      }
   }
   return maxlen;
   }
```

Brute Force II: $TC = O(N^2)$ and $SC = O(1)$

- Run two loops i and j, do the prefixsum and check if the sum ==0 or not, if sum is 0, calculate its length, take maximum of current length and previous calculated length.

i=0

	currsum
j=0	{15}
j=0 to j=1	{13}
j=0 to j=2	{15}
j=0 to j=3	{7}
j=0 to j=4	{8}
j=0 to j=5	{15}
j=0 to j=6	{25}
j=0 to j=7	{48}

i=1

	currsum
j=1 to j=1	{-2}
j=1 to j=2	{0}
j=1 to j=3	{-8}
j=1 to j=4	{-7}
j=1 to j=5	{0}
j=1 to j=6	{ 10}
j=1 to j=7	{33}

i=2

	currsum
j=2 to j=2	{2}
j=2 to j=3	{-6}
j=2 to j=4	{-5}
j=2 to j=5	{2}
j=2 to j=6	{8}

i=3

	currsum
j=3 to j=3	{-8}
j=3 to j=4	{-7}
j=3 to j=5	{0}
j=3 to j=6	{10}

j=2 to j=7	{31}

j=3 to j=7	{33}

i=4

	currsum
j=4 to j=4	{1}
j=4 to j=5	{8}
j=4 to j=6	{18}
j=4 to j=7	{41}

i=5

	currsum
j=5 to j=5	{7}
j=5 to j=6	{17}
j=5 to j=7	{40}

i=6

	currsum
j=6 to j=6	{10}
j=6 to j=7	{33}

i=7

	currsum
j=7 to j=7	{23}

- *There are 3 subarrays with 0 sum, in which the largest subarray length is 5.*

```
1  int maxLen(vector<int>&arr, int n)
2      {    int max_len = 0;
3
4
5      for (int i = 0; i < n; i++) {
6
7          int curr_sum = 0;
8
9          for (int j = i; j < n; j++) {
10             curr_sum += arr[j];
11
12             if (curr_sum == 0)
13                 max_len = max(max_len, j - i + 1);
14         }
15     }
16     return max_len;
17     }
```

Optimal Approach : TC= O(N) and SC = O(N)

PrefixSum	index
0	-1
15	0
13	1
7	3
8	4
25	6
48	7

A[] = {15, -2, 2, -8, 1, 7, 10, 23}

X + K = X

means, K is 0, so if the X value is repeating in a map means there is a subarray in between whose sum is 0. (k is a subarray)

- We will store the prefix sum in the map, if map already contains that value, means we got a subarray, whose sum is 0.
- To find out its length, we will do i-map[PrefixSum].
  Here 15 is repeating at index 2 and index 5, so length will be 2-0=2 and 5-0 =5. 5 is maximum so it will store in the map.

```cpp
int maxLen(vector<int>&a, int n)
    {

    int sum=0,ans=0;
    unordered_map<int,int> m;
    m[0]=-1;
    for(int i=0;i<n;i++){
        sum+=a[i];
        if(m.find(sum)!=m.end()){
            ans = max(ans,i-m[sum]);
        }else m[sum] = i;

    }
    return ans;
    }
```

- *Do we need to check explicitly for zero or it can be handled?*

Answer: yes, we need to store map[0] = -1,

arr = {0, 1, -1, 0}

Prefixsum	index
0	-1
1	1

i=0, prefixsum =0, i-prefixsum = 0-(-1) = 1

i=1, prefixsum =1, map[1] = 1

i=2, prefixsum = 0. Length = 2 - (-1) = 3

i=3, prefixsum = 0, Length = 3 - (-1) = 4

Problem-4 HashMap

31 December 2021 21:00

Problem 4 : Count Of All Subarrays With Zero Sum

Input:	N = 8 A[] = {15,-2,2,-8,1,7,10,23}
Output:	3

NOTE : This problem is same as above one, Just increase count instead of calculating length.

Problem Statement : *we have to count the subarrays having 0 sum.*
{-2,2} , sum = 0,
{-8,1,7}, sum = 0,
{-2,2,8,1,7}, sum = 0.
There are 3 subarrays, so output will be 3.

Brute Force I: *TC = $O(N^3)$ and SC = $O(1)$*

```
int maxLen(vector<int>&A, int n)
    {    int count = 0;

    for(int i = 0; i<n; i++)
    {
       for(int j = i; j<n;j++)
       {
            int sum = 0;
            for(int k = i; k<=j; k++)
                sum = sum + A[k];
            if(sum == 0)
            {
              count++;
            }
       }
    }
    return count;
    }
```

Brute Force II: TC = $O(N^2)$ and SC = $O(1)$

LinkedIn

```cpp
int maxLen(vector<int>&A, int n)
    {    int count = 0;

    for(int i = 0; i<n; i++)
    {
       for(int j = i; j<n;j++)
       {
            int sum = 0;
            for(int k = i; k<=j; k++)
                sum = sum + A[k];
            if(sum == 0)
            {

               count++;
            }
       }
    }
    return count;
    }
```

Optimal Approach TC= $O(N)$ and SC = $O(N)$

A[] = {15, -2, 2, -8, 1, 7, 10, 23}

at i=0, count of 15 is 1
i=2, count of 15 is 2
at i=4, count of 15 is 3.

PrefixSum	Count
0	1
15	~~1~~ 2 3
13	1
7	1
8	1
25	1
48	1

NOTE: To handle, zero sum subarray, either store m[0]=1, or use if(sum==0) count++.

```cpp
ll findSubarray(vector<ll> arr, int n ) {

        unordered_map<int,int> map;
        int sum=0;
        int count=0;
        for(int i=0;i<n;i++){
            sum+=arr[i];
            if(sum==0){
                count++;
            }
            if(map.find(sum)!=map.end()){
                count+=map[sum];
            }
            map[sum]++;
        }
        return count;
    }
```

Problem - 5, HashMap

31 December 2021 21:00

Problem 5 : Count distinct elements in every window (imp)

Problem Statement :

Given an array of integers and a number K. Find the count of distinct elements in every window of size K in the array.

Input:	N = 7, K = 4 A[] = {1,2,1,3,4,2,3}
Output	3 4 4 3

Subarray	Distinct elements
{1,2,1,3}	3
{2,1,3,4}	4
{1,3,4,2}	4
{3,4,2,3}	3

Brute Force : TC - O(klogk * (N-k)), SC - O(k + (N- k +1))

- Use 2 for loops, outer loop from i= 0 till i<=n-k, because last window will start from index n-k.
- inner loop from j=i to j<i + k, (basically till the last element)
- Store the elements in the set.
- Calculate the size of the set and store it in the resultant vector.

NOTE : Set only stores distinct elements, so set size will give count of distinct elements.

```
vector <int> countDistinct (int arr[], int n, int k)
    {
      vector<int> res;

        for(int i = 0; i <=n-k; i++)
        {    int dist_count = 1;
            set<int> s;
          for(int j = i; j<i+k; j++)
          {
              s.insert(arr[j]);
          }
            int size= s.size();
          res.push_back(size);
        }

        return res;
    }
```

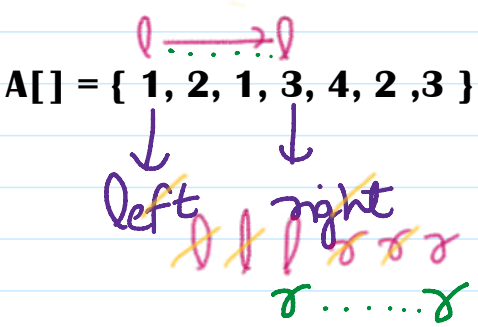
- Since We are using two loops, so time complexity of loops will be (n-k) * k and inserting an element in a set takes logk time, so total TC = (n-k)*klogk.
- We will store at max k elements in the set, Resultant vector will be of size atmax O(n-k+1) .so SC = O(K + n-k+1).

But we can ignore the space used for resultant vector, because it is requirement of the program.

Optimal Approach : TC - O(N), SC - O(N)

A[ ] = { 1, 2, 1, 3, 4, 2 ,3 } **N=7, k=3**

- Store first k elements in the map, calculate the map size and store it in resultant vector.
  (This will be done for first window, map size will return no of distinct elements).
- From second window, we will maintain two pointers, left and right, left pointer will start from 0 and right pointer will start from k



Element	Count
1	2 1 0
2	1 0 1
3	1
4	1

- Left = 0, right = k, now we will decrease the count of arr[left] from map, and increase the count of arr[right] in map.
  So now, map will have **{2,1,3}** store the map size in resultant vector. Increament Left.
- left = 1, right =k+1,
  arr[left] = 2, arr[right] = 4, so store arr[4] in map. Decrease the count of arr[left] from map.
  So now, map will have **{1,3,4}** store map size in resultant vector. Increament left.
- left = 2, right = K+2, arr[left] = 1, arr[right] = 2, so store arr[2] in map. Decrease the count of arr[left] from map.
  arr[left] =1, count of 1 is 0, so delete element 1 from map.
  Now, map will have {3,4,2} store map size in resultant vector. Increament left.
- left = 3, right = K+3,arr[left]=3, arr[right]= 3, store arr[4] in map. Decrease the count of arr[left].

```cpp
vector <int> countDistinct (int arr[], int n, int k)
    {

        unordered_map<int, int> hashMap;

        for(int i = 0; i < k; i++) {
            hashMap[arr[i]]++;
        }

        vector<int> res;
        res.push_back(hashMap.size());
        int left = 0;

        for(int right = k; right < n; right++) {
            hashMap[arr[right]]++;
            hashMap[arr[left]]--;

            if(hashMap[arr[left]] == 0) {
                hashMap.erase(arr[left]);
            }

            left++;

            res.push_back(hashMap.size());
        }

        return res;

    }
```

Problem 6, HashMap

31 December 2021 20:55

Problem 6 : Longest Substring Without Repeating Characters

Problem statement : Given a string s, find the length of the longest substring without repeating characters.

Input:	s = "abcabcbb"
Output:	3
Explanation:	The answer is "abc", with the length of 3.

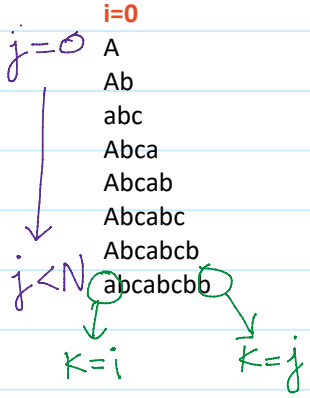
LinkedIn in.linkedin.com/in/kapilyadav22

Brute Force : TC:$O(N^3)$, SC: $O(1)$

S : - 'abcabcbb'

- Iterate through all the substrings and check if they contain duplicates or not.
- Run i loop from 0 to N, and j loop to find substring.
- Run k loop to iterate over every substring.

All the possible substrings are :



i=0
A
Ab
abc
Abca
Abcab
Abcabc
Abcabcb
abcabcbb

i=1
B
bc
bca
bcab
bcabc
bcabcb
bcabcbb

i=2
c
ca
cab
cabc
cabcb
cabcbb

i=3
a
ab
abc
abcb
abcbb

i=4
b
bc
bcb
bcbb

i=5
c
cb
cbb

i=6
b
bb

i=7
b

```cpp
int longestUniqueSubsttr(string str){
        int n = str.size();
    int res = 0;
    for (int i = 0; i < n; i++)
        {
            for (int j = i; j < n; j++)
            {     vector<bool> visited(26);
             for (int k = i; k <= j; k++)
                {
                if (visited[str[k] - 'a'] == true)
                        break;
                visited[str[k] - 'a'] = true;
                res = max(res, k - i + 1);
                }
            }
        }
    return res;

    }
```

Better Approach - $TC : O(N^2), SC : O(1)$

- Use sliding window approach
- Here we will not find all the substrings, we will iterate till the distinct element for every i iteration.
  for i=0 and j=3 , a is repeating, so further substrings will also contain repeating elements, hence break the loop and don't consider them.

S : - 'abcabcbb'

i=0	i=1	i=2
a	b	c
ab	bc	ca
abc	bca	cab

i=3	i=4	i=5	i=6	i=7
a	b	c	b	b
ab	bc			
abc				

```cpp
int longestUniqueSubsttr(string str){
        int n = str.size();
    int res = 0; // result

    for (int i = 0; i < n; i++) {

        vector<bool> visited(256);

        for (int j = i; j < n; j++) {

            if (visited[str[j]] == true)
                break;

            else {
                res = max(res, j - i + 1);
                visited[str[j]] = true;
            }
        }

        visited[str[i]] = false;
    }
    return res;
    }
```

Optimal approach : *use Hashmap TC : O(N), SC : O(N)*

s : - 'abcabcbb'

- right=0, str[0]=a, memo[a]=0, len = 1 , answer = 1
- right=1, str[1]=b, memo[b]=1, len = 2, answer = 2
- right=2, str[2]=c, memo[c]=1, len = 3, answer = 3
- right=3, str[3]=a, left = mem[a]+1 =0+1 = 1, len = 3, answer = 3 , memo[a]=3
- right=4, str[4]=b, left= memo[b]+1 = 2, len = 2, answer = 3,memo[b]= 4
- right = 5, str[5]= c, left= memo[c]+1 = 3, len = 2, answer = 3 memo[c] = 5
- right = 6, str[6]= b, left= memo[b]+1 = 5, len = 1, answer = 3 memo[b] = 6
- right = 7, str[7]= a, left= memo[b]+1 = 7, len = 1, answer = 3 memo[b] = 7

Char	Index
a	~~0~~ 3
b	~~1~~ 4 6 7
c	~~2~~ 5

Example -2: S = {abaabc}

Char	Index
a	~~0~~ 3
b	~~1~~ 4
c	5

- right=0, str[0]=a, memo[a]=0, len = 1 {a}, answer = 1
- right=1, str[1]=b, memo[b]=1, len = 2 {a, b} , answer = 2
- right=2, str[2]=a, left = mem[a]+1 =0+1 = 1, len = 2 {b, a}, answer = 2 , memo[a]=2
- right=3, str[3]=a, left= memo[a]+1 = 3, len = 1 {a} , answer = 2, memo[a]= 3
- Here is the interesting case:

Now left is 3, but our b's last index was 1, so memo[b]<left,
so don’t update the left.
right =4, str[4]= b, len = 2 {a ,b} , answer = 2 memo[b] = 4
right=5, str[5]=c, len = 3 {a, b, c} , answer =3, memo[a]= 3

```
int lengthOfLongestSubstring(string s) {
        int answer=0;
        int left=0;
        int len=0;

        unordered_map<char,int> memo;

        int m=s.size();
        for(int right=0; right<m; right++)
        {
            char currentchar = s[right];

            if(memo.find(currentchar)!=memo.end()  &&
memo[currentchar]>=left)
            {
                left= memo[currentchar]+1;
            }
            len =right-left+1;
            memo[currentchar] = right;
            answer = max(answer,len);
        }
    return answer;
    }
```

Problem- 7, HashMap

01 January 2022 15:19

Problem - 7 : Substrings with exactly K distinct chars

Given a string you need to print the size of the longest possible substring that has **exactly K** unique characters. If there is no possible substring then print -1.

Input:	S = "aabacbebebe", K = 3
Output:	7
Explanation:	"cbebebe" is the longest substring with K distinct characters.

Brute Force : *TC:$O(N^3)$, SC: $O(1)$*

- Generate all the subarrays and check if the count ==k or not.
  (Same as Problem 6, Brute force approach)

```
int longest(string str, int k)
{
    int n = str.size();
    int res = -1;
 for(int i=0;i<n;i++)
 {
     for(int j=i; j<n; j++)
     {    int count=0;
           int m;
         vector<bool> visited(26);
         for(m=i; m<=j; m++)
         {
             if(visited[str[m]-'a']==true)
                  continue;
            visited[str[m]-'a']=true;
            count++;

         }
         if(count==k)
         {
             res = max(res,m-i);
         }

     }
 }
return res;
}
```

Better Approach - $TC : O(N^2), SC : O(1)$

- In this approach, we are skipping repeating elements and counting distinct elements and if(count==k) calculating the length of substring.
- Visited array is used to maintain distinct count.

```
    int longestKSubstr(string str, int k) {
    // your code here
        int n = str.size();
    int res = -1;

for(int i=0;i<n;i++)
{  int j;
 vector<bool> visited(26);
 int count=0;
    for(j=i; j<n; j++)
    {
            if(visited[str[j]-'a']!=true)
            {  visited[str[j]-'a']=true;
            count++;
            }
    if(count==k)
        {
            res = max(res,j-i+1);
        }
    }

}
return res;
    }
```

Optimal : Using Hashmap: TC: O(2N) = O(N), SC: O(K)

S = aabacbebebe

Char	Count
a	~~1~~ ~~2~~ ~~3~~ 2 X 0
b	~~1~~ ~~2~~ 3
c	1
e	~~1~~ ~~2~~ 3

- Insert characters in map.
  - If map size<k, just insert char in map.
  - If map size>k, slide the window, by decreasing the count of map[left].
  - If map[left] count is 0, remove the char from map.
  - If map size=k, calculate the length.

So, at last map contains b, c, e with frequencies 3,1 and 3. So output will be 7.

```cpp
int longestKSubstr(string s, int k) {

    int answer= -1;
    int left=0;
    unordered_map<char,int> memo;
    int m=s.size();
    for(int right=0;right<m;right++)
    {
        memo[s[right]]++;

        while(left<=right && memo.size()>k)
        {
            memo[s[left]]--;

            if(memo[s[left]]==0)
                memo.erase(s[left]);
             left++;

        }
        if(memo.size()==k)
           answer= max(answer,right-left+1);

    }
    return answer;
    }

```

LinkedIn in.linkedin.com/in/kapilyadav22

Problem-8, HashMap

01 January 2022 18:28

Problem-8: [Count number of substrings with exactly k distinct characters]

Exactly k distinct elements can found as :
Substrings having at most k distinct characters - Substrings having at most k-1 distinct characters

Optimal : Using Hashmap: TC: O(N), SC: O(N)

```cpp
int subarraysWithKDistinct(vector<int>& s, int k) {
        return countatmostk(s,k) - countatmostk(s,k-1);

    }
   int countatmostk(vector<int>& str, int k)
{   int size=str.size();
    int dist_count=0;
    int res=0;
    int release=0;

    unordered_map<int,int> map;
    for(int acquire = 0; acquire < size; acquire++)
    {    int curr_char = str[acquire];

        if(map.find(curr_char)==map.end())
            dist_count++;
        map[curr_char]++;

        while(release<=acquire && dist_count>k)
        {
            int dis_char=str[release];
            map[dis_char]--;

             if(map[dis_char]==0)
               {
                    map.erase(dis_char);
                    dist_count--;
               }
             release++;
        }
        res+=(acquire-release+1);
 }
return res;
}
```

Problem-9, HashMap

01 January 2022 18:48

Problem -9: Longest subarray with equal number of 0 and 1.

Input:	nums = [0,1,0,1,0]
Output:	4
Explanation:	[0, 1,0,1] is the longest contiguous subarray with an equal number of 0 and 1

Brute Force : *TC:$O(N^3)$, SC: $O(1)$*

Same as Problem-6, iterate through all the subarrays and check whether count of 0s and 1s are equal or not

```
int maxLen(int nums[], int n)
    {

    int res = 0;
 for(int i=0;i<n;i++)
 {
     for(int j=i; j<n; j++)
     {
           int m;
         vector<int> v(2);
         for(m=i; m<=j; m++)
         {
            v[nums[m]]=v[nums[m]]+1;
         }
         if(v[0]==v[1])
         {
             res = max(res,m-i);
         }
         // cout<<v[0]<<" "<<v[1]<<endl;

     }
 }

return res;
    }
```

Better Approach - TC : O(N²), SC : O(1)

- Use 2 for loops, and compare if the count of 0s is equal to count of 1 or not.

```
int maxLen(int nums[], int n)
    {

    int res = 0;
     for(int i=0;i<n;i++)
     { int j;
     vector<int> v(2);
         for( j=i; j<n; j++)
         {
            v[nums[j]]=v[nums[j]]+1;
             if(v[0]==v[1])
             {
                 res = max(res,j-i+1);
             }
         }
 }

return res;
    }
```

Optimal Approach : TC - O(N), SC - O(N)

Example - [0,1,0,1,1]

Sum	index
0	-1
-1	0
1	4

- i=0; 0 is in map, sum = -1, ans = 0.
- i=1, 1 is not in map, sum=-1+1 = 0, ans= 1-(-1) = 2
- i=2, 0 is in map, sum= 0-1, ans=2
- i=3, 1 is in map, sum = 0, ans= 3-(-1) = 4 ,
- i=4, 1 is in map, sum =-1, ans = max(ans, 4 - 4) =4

```cpp
int maxLen(int nums[], int m)
   {
        int sum=0,ans=0;
   unordered_map<int,int> map;
   map[0]=-1;
   for(int i=0;i<m;i++){
       if(nums[i]==0)
           sum+=-1;
       else
       sum+=nums[i];
       if(map.find(sum)!=map.end()){
           ans = max(ans,i-map[sum]);
       }
       else map[sum] = i;
   }
   return ans;
   }
```

Follow up : find the start and end index of elements having equal number of 0s and 1s.

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> maxLen(int nums[], int m)
    {
         int start = -1;
         int end = -1;
         int sum=0;
         int ans=0;
         vector<int> v;
    unordered_map<int,int> map;
    map[0]=-1;
    for(int i=0;i<m;i++){
        if(nums[i]==0)
            sum+=-1;
        else
        sum+=nums[i];
        if(map.find(sum)!=map.end()){
            int temp = i-map[sum];
            if(temp>ans)
            {  ans= temp;
               start= map[sum]+1;
               end = i;
            }
        }
        else map[sum] = i;
    }
    v.push_back(start);
    v.push_back(end);
    return v;

    }

    int main()
{
         int n;
         cin >> n;
         vector<int> res;
         int a[n];
         for (int i = 0; i < n; i++)
             cin >> a[i];

         res = maxLen(a, n);
         cout<<res[0]<<" "<<res[1]<<endl;

    return 0;
}
```

Problem-10, HashMap

02 January 2022 11:39

Problem-10 : Subarrays with equal 1s and 0s

- Same as **Zero Sum Subarrays.**
- we just need to put the check, if arr[i]==0, sum+=-1, else sum+=1.

Optimal Approach : TC - O(N), SC - O(N)

```
long long int countSubarrWithEqualZeroAndOne(int arr[], int n)
    {

        unordered_map<int,int> map;
         int sum=0;
         int count=0;
         map[0]=1;
         for(int i=0;i<n;i++){
            if(arr[i]==0)
               sum+=-1;
             else
              sum+=1;

             if(map.find(sum)!=map.end()){
                 count+=map[sum];
             }
             map[sum]++;
         }
         return count;
    }
```

Problem - 11, HashMap

02 January 2022 13:22

Problem -11 : Longest Subarray With Equal Number Of 0s 1s And 2s

arr[] = {0,1,2,2,1,0,2,2,1,0}

Brute Force : TC:O(N^3), SC: O(1)

- Iterate through all the subarrays and check whether count of 0s , 1s and 2s are equal or not.

NOTE: I am using vector of size 3 to store the count of 0, 1 and 2, we can use 3 variables to store the count.

```
int maxLen(int nums[], int n)
    {

    int res = 0;
 for(int i=0;i<n;i++)
 {
     for(int j=i; j<n; j++)
     {
           int m;
         vector<int> v(3);
         for(m=i; m<=j; m++)
         {
            v[nums[m]]=v[nums[m]]+1;
         }
         if((v[0]==v[1]) && (v[1]==v[2]))
         {
             res = max(res,m-i);
         }
     }
 }

return res;
    }
```

Better Approach - TC : O(N^2), SC : O(1)

- Use 2 for loops, and compare if the count of 0s , 1s and 2s are equal or not.

```cpp
int maxLen(int nums[], int n)
    {
    int res = 0;
 for(int i=0;i<n;i++)
 { int j;
 vector<int> v(3);
     for( j=i; j<n; j++)
     {
        v[nums[j]]=v[nums[j]]+1;
         if(v[0]==v[1] & v[1]==v[2])
         {
             res = max(res,j-i+1);
         }
     }
 }

return res;
    }
```

Optimal Approach : TC - O(N), SC - O(N)

arr[] = {0, 1, 2, 2, 1, 0, 2, 2, 1, 0}

- Make pair of (countones-countzero) and (counttwos-countones), Use it as a key in map, store index in value.
- Iterate through the array, if map already contains the value, means we got our subarray, having equal number of 0,1 and 2.

Key	Index
0 0	-1
-1 0	0
0 -1	1
0 1	3
1 0	4
0 2	7
1 1	8

Countzero = 0 1 2 3
Countones = 0 1 2 3
Counttwos = 0 1 2 3 4

DRY RUN:

i	arr[i]	countzero	countones	counttwos	ountones-countzero	counttwos-countones	Map Value (INDEX)	ans
0	0	1	0	0	-1	0	0	0
1	1	1	1	0	0	-1	1	0
2	2	1	1	1	0	0	-1	=2-(-1) = 3
3	2	1	1	2	0	1	3	3
4	1	1	2	2	1	0	4	3
5	0	2	2	2	0	0	-1	= max(3,6) =6.
6	2	2	2	3	0	1	3	= max(3,6) =6
7	2	2		4	0	2	7	6
8	1	2	3	4	1	1	1	6
9	0	3	3	4	0	1	3	= max(6,6) =6

```cpp
#include<bits/stdc++.h>
using namespace std;

int solve(vector<int>&nums)
{
    int ans = 0;
     map <pair <int,int>, int> mp;
     int countZero=0,countOnes=0,countTwos=0;

     mp[make_pair(0,0)]=-1;
     for(int i=0;i<nums.size();i++)
     {
         if(nums[i]==0)
           countZero+=1;
         else if(nums[i]==1)
            countOnes+=1;
         else
           countTwos+=1;

   pair <int, int> currkey= make_pair(countOnes-countZero,countTwos-countOnes);

       if(mp.find(currkey)!=mp.end())
       {
           int prevIndex = mp[currkey];
           int length = i-prevIndex;
            ans=max(ans,length);
       }
     else
       mp[currkey]=i;
     }
     return ans;
}

int main()
{
     int n;
     cin>>n;
     vector<int>nums(n);
     for(int i=0;i<n;i++)
     {
         cin>>nums[i];
     }
     cout<<solve(nums);
     return 0;
}
```

LinkedIn in.linkedin.com/in/kapilyadav22

Problem-12, HashMap

02 January 2022 20:31

Problem -12 : Equal 0, 1 and 2

Brute force and better approaches are same as problem 11,
So lets discuss optimal approach. (Chaliye shuru Karte Hain 🤷)

Key	Frequency
0 0	~~1~~ ~~2~~ 3
-1 0	1
0 -1	1
0 1	~~1~~ 2
1 0	1
0 2	1
1 1	1

Optimal Approach : TC - O(N), SC - O(N)

- Make pair of (countones-countzero) and (counttwos-countones), Use it as a key in map, store frequency in value.
- Iterate through the array, if map already contains the value, means we got our subarray, having equal number of 0,1 and 2, so

arr[] = {0, 1, 2, 2, 1, 0, 2, 2, 1, 0}

DRY RUN:

i	arr[i]	countzero	countones	counttwos	countones-countzero	counttwos-countones	Map Status	ans
0	0	1	0	0	-1	0	store 1 in the map.	0
1	1	1	1	0	0	-1	store 1 in the map	0
2	2	1	1	1	0	0	already in the map, increase frequency.	0+1
3	2	1	1	2	0	1	store 1 in the map	1
4	1	1	2	2	1	0	store 1 in the map	1
5	0	2	2	2	0	0	already in the map, increase frequency.	1+2=3
6	2	2	2	3	0	1	already in the map, increase frequency.	3+1=4
7	2	2		4	0	2	store 1 in the map	
8	1	2	3	4	1	1	store 1 in the map	
9	0	3	3	4	0	1	already in the map, increase frequency.	**4+2=6**

```cpp
long long getSubstringWithEqual012(string nums) {
        int ans = 0;
    map <pair <int,int>, int> mp;
    int countZero=0,countOnes=0,countTwos=0;

    mp[make_pair(0,0)]=1;
    for(int i=0;i<nums.size();i++)
    {
        if(nums[i]=='0')
          countZero+=1;
        else if(nums[i]=='1')
           countOnes+=1;
        else
          countTwos+=1;

  pair <int, int> currkey= make_pair(countOnes-countZero,countTwos-countOnes);

      if(mp.find(currkey)!=mp.end())
      {
          ans+= mp[currkey];
    }
    mp[currkey]++;
    }
    return ans;
    }
```

LinkedIn in.linkedin.com/in/kapilyadav22

Problem-13, Hashmap

02 January 2022 21:39

Problem 13: Subarray Sum Equals K

Brute Force : TC:O(N³), SC: O(1)

Generate all the subarray and check if count==k or not.

```cpp
int countSubarraysWithSumK(vector <int> & a, int K) {
  int n = a.size();
  int count = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      int sum = 0;
      for (int k = i; k <= j; k++) {
        sum += a[k];
      }
      if(sum==K)
       count++;
    }
  }
  return count;
}
```

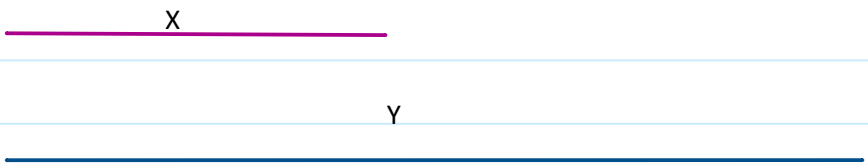
Better Approach - TC : O(N²), SC : O(1)

Use 2 for loops, and compare if the count of 0s, 1s and 2s are equal or not.

```cpp
int countSubarraysWithSumK(vector <int> & arr, int K) {
  int res=0;
  int n=arr.size();
  for (int i = 0; i < n; i++)
  {
    int sum = 0;
    for (int j = i; j < n; j++)
    {
      sum += arr[j];

      if (sum == K)
        res++;
    }
  }
  return res;
}
```

Optimal Approach : TC - O(N), SC - O(N)



If Y-K =X, and X is already present in the map, means we got a subarray equal to k.
Lets say : X = 2, Y=5,K=3, So 5-3=2, which is present in the map, so we got one subarray sum equal to k.

Example : arr[] = {2, 1, 2, 2, 1, 2, 1, 1}
k = 3

i	Arr[i]	Prefixsum	Prefixsum-k	Ans	Map status
0	2	2	2-3=-1	0	Store 1 in map [2]
1	1	3	3-3=**0**	1	Store 1 in map[3]
2	2	5	5-3=**2**	2	Store 1 in map[5]
3	2	7	7-3=4	2	Store 1 in map[7]
4	1	8	8-3=**5**	3	Store 1 in map[8]
5	2	10	10-3=**7**	4	Store 1 in map[10]
6	1	11	11-3=**8**	5	Store 1 in map[11]
7	1	12	12-3=9	**5**	Store 1 in map[12]

PrefixSum	Frequency
0	1
2	1
3	1
5	1
7	1
8	1
10	1
11	1
12	1

```cpp
int subarraySum(vector<int>& nums, int k) {
     int answer=0;
     int prefixsum =0;

    unordered_map<int,int> memo;
    memo[0]=1;
    int n=nums.size();
    for(int i=0;i<n; i++)
    {  prefixsum+=nums[i];

        if(memo.find(prefixsum-k)≠memo.end())
        {
            answer+=memo[prefixsum-k];
        }
        memo[prefixsum]++;
     }
        return answer;
    }
```

Q. Why we are storing map[0]=1?

Ans : Lets say we have arr[] = {2,3,0} and k=5, then there will be two subarray whose sub is equal to k. {2,3} and {2,3,0}. Storing 0 in map, so that as soon as we encounter the condition, where there is a 0 in the array, we will increase our count.

NOTE : we can do the same thing using if condition in the loop.

LinkedIn in.linkedin.com/in/kapilyadav22

Problem-14, HashMap

03 January 2022 19:09

Problem-14: Subarray Sums Divisible by K

Problem Statement: We have given an array, we need to find out the number of subarrays whose sum is divisible by k.

Example 1 :

Input	nums = [4,5,0,-2,-3,1], k = 5
Output	7
Explanation	[4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3] are the subarrays which are divisible by K.

Brute Force : TC:$O(N^3)$, SC: $O(1)$

- Generate all the subarray and check if sum is divisible by k or not.

```
int countSubarraysdivisiblebyK(vector <int> & a, int K) {
  int n = a.size();
  int count = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      int sum = 0;
      for (int k = i; k <= j; k++) {
        sum += a[k];
      }
      if(sum%K==0)
       count++;
    }
  }
  return count;
}
```

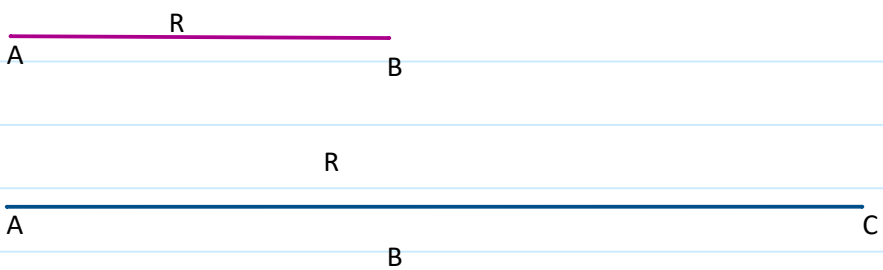
Better Approach - TC : $O(N^2)$, SC : $O(1)$

- We are using ***currentkey = (((a[j])%K)+K)%K;*** to handle negative numbers.
- Since remainder cannnot be in negative, so we can simply do ***prefixremainder = sum%K;*** to find the remainder.

```cpp
int countSubarraysdivisiblebyK(vector <int> & a, int K) {
  int n = a.size();
  int count = 0;
  for (int i = 0; i < n; i++)
  {
      int sum = 0;
    for (int j = i; j < n; j++)
    {
        int currentkey = (((a[j])%K)+K)%K;
        sum+=currentkey;
        int prefixremainder = sum%K;

        if(prefixremainder==0)
           count++;
    }
  }
  return count;
}
```

Optimal Approach : TC - $O(N)$, SC - $O(N)$



- If in a subarray A to B, the remainder is R, and in subarray A to C, the remainder is again R, we can say that, B to C is divisible by K.

Example = {4,5}, k=5, at index 0, remainder will be 4, again at index 1, the remainder will be 4+5 =9%5=4, so we can say that we got our first subarray, which is divisible by k=5.

Example : arr = [4,5,0,-2,-3,1], k = 5

i	Arr[i]	PrefixRem	Ans	Map status
0	4	4	0	Store 1 in map [4]
1	5	4	1	Map[4]++
2	0	4	=1+2=3	Map[4]++
3	-2	2	3	Store 1 in map[2]
4	-3	4	=3+3=6	Map[4]++
5	1	0	7	Map[0]++

PrefixRem	Frequency
0	~~1~~ 2
4	~~1~~ ~~2~~ ~~3~~ 4
2	1

- Store map[0]=1, Reason we have already discussed in problem 13

```cpp
int subarraysDivByK(vector<int>& nums, int k) {
        int answer =0;
         int prefixRem = 0;
         unordered_map<int,int> map;

         int m=nums.size();
         map[prefixRem]=1;

         for(int i=0;i<m;i++)
         {
             prefixRem+=nums[i];
             prefixRem=((prefixRem%k)+k)%k;

             if(map.find(prefixRem)!=map.end())
             {     answer+=map[prefixRem];
             }
             map[prefixRem]++;

         }
     return answer;
     }
```

Problem-15, HashMap

03 January 2022 20:39

Problem-15 : Longest subarray with sum divisible by K

Problem Statement: We have given an array, we need to find out longest subarray whose sum is divisible by k.

Example 1:

Input:	A[] = {2, 7, 6, 1, 4, 5} K = 3
Output:	4
Explanation:	The subarray is {7, 6, 1, 4} with sum 18, which is divisible by 3.

Brute Force : TC:$O(N^3)$, SC: $O(1)$

Generate all the subarray and check if sum is divisible by k or not and if yes, find maximum of length of subarrays whose sum is divisible by k.

```
int longSubarrWthSumDivByK(int a[], int n, int K)
    {
          int ans = 0;
          int k;
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      int sum = 0;
      for (k = i; k <= j; k++) {
        sum += a[k];
      }
      if(sum%K==0)
       ans=max(ans,k-i);
    }
  }
  return ans;

    }
```

Better Approach - TC : $O(N^2)$, SC : $O(1)$

- Run 2 loops i and j,and check if sum is divisible by k or not and if yes, find maximum of length of subarrays whose sum is divisible by k.

```cpp
int longSubarrWthSumDivByK(int a[], int n, int K)
    {
        int ans = 0;
  for (int i = 0; i < n; i++)
  {
      int sum = 0;
    for (int j = i; j < n; j++)
    {
        int currentkey = (((a[j])%K)+K)%K;
        sum+=currentkey;
      int prefixremainder = sum%K;
       if(prefixremainder==0)
           ans=max(ans,j-i+1);
    }
  }
  return ans;

    }
```

Optimal Approach : TC - O(N), SC - O(N)
Store the index of prefixremainder in map, and compare the length, store max length in answer.

A[] = {2, 7, 6, 1, 4, 5} K = 3

i	Arr[i]	PrefixRem	i-map[prefixrem]	Ans	Map status
0	2	2	Not in map	0	Store 1 in map [2]
1	7	0	=1-(-1)=2	2	Already in map
2	6	0	=2-(-1)=2	3	Already in map
3	1	1	Not in map	3	Store 3 in map[1]
4	4	2	=4-0=4	4	Already in map
5	5	1	=5-3=2	=max(2,4) =4	Already in map

PrefixRem	Index
0	-1
2	0
1	3

- Store map[0]=-1, because if we get 0 in subarray, so 0 is divisible so we increase the length .
- Ex - Arr = {5,0}, k = 5, so arr[1]=0, so length = 1-(-1)=2.

```cpp
int longSubarrWthSumDivByK(int nums[], int n, int k)
    {
         int answer =0;
        int prefixRem = 0;
        unordered_map<int,int> map;
        map[0]=-1;

        for(int i=0;i<n;i++)
        {
            prefixRem+=nums[i];
            prefixRem=((prefixRem%k)+k)%k;


            if(map.find(prefixRem)!=map.end())
            {     answer=max(answer,i-map[prefixRem]);

            }
            else
                map[prefixRem] = i;

        }
    return answer;

    }
```

Problem-16, HashMap

04 January 2022 13:12

Problem-16: Valid Anagram

Problem Statement :
Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Input:	s = "anagram", t = "nagaram"
Output:	true

Brute Force : Use Sorting:

TC - O(NLogN) where N is the size of each string,
SC: O(1)no extra space (sorting algos may take some space)

```
bool areAnagram(string str1, string str2)
{  //brute force sorting
    int s1 = str1.length();
    int s2 = str2.length();

    if (s1 ≠ s2)
        return false;


    sort(str1.begin(), str1.end());
    sort(str2.begin(), str2.end());

    for (int i = 0; i < s1; i++)
        if (str1[i] ≠ str2[i])
            return false;

    return true;
}
```

Optimal Approach : *TC = O(N), SC - O(1), because there are constant unique characters.*

Input:	s = "anagram", t = "nagaram"

Char	frequency
a	1 2 3 2 1 0
n	1 0
g	1 0
r	1 0
m	1 0

```cpp
bool isAnagram(string s, string t) {

    if (s.size()!=t.size())
        return false;

    unordered_map <char, int> umap;

    for (char c:s)
        umap[c]++;

    for(char c:t) {
        if(umap.find(c)==umap.end())
            return false;
        umap[c]--;
        if(umap[c]<0)
        return false;
    }
    return true;
}
```

Using Array:

```cpp
bool isAnagram(string s, string t)
{
         if (s.size() != t.size())
            return false;

        int letterTable[26]={0};

        for (int i = 0; i < s.size(); i++)
            letterTable[s[i] - 'a']++;
        for (int i = 0; i < t.size(); i++)
        {
            if (letterTable[t[i] - 'a'] == 0)
                return false;
            letterTable[t[i] - 'a']--;
        }
        return true;
    }
```

Problem-17 Group Anagrams

04 January 2022 13:48

Problem-17: Group Anagrams

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

Input:	strs = ["eat","tea","tan","ate","nat","bat"]
Output:	[["bat"],["nat","tan"],["ate","eat","tea"]]

- *Sort each string and store the anagrams in the map of that string.*

String	Vector<string>
aet	{eat, ate, tea}
ant	{nat, tan}
abt	{bat}

Time Complexity :

O(n.klog(k)) where k is the length of string and n is total no. of strings

Space Complexity :

O(n) if none of the strings are anagram.

```cpp
vector<vector<string>> groupAnagrams(vector<string>& strs) {
        vector<vector<string>> ans;
        unordered_map<string, vector<string>> map;

        for(int i = 0; i < strs.size(); i++){
            string temp = strs[i];

            sort(strs[i].begin(), strs[i].end());
            map[strs[i]].push_back(temp);
        }

        for(auto it = map.begin(); it != map.end(); it++)
{
            ans.push_back(it->second);
        }

        return ans;
    }
```

Problem-18, HashMap

04 January 2022 18:29

Problem -18 : Count Of Substrings Having At Most K Unique Characters

Input:	aabca, k=2
Output:	10

Substrings having at most 2 unique characters are:

'a' 'a' 'aa' 'ab' 'aab' 'b' 'bc' 'c' 'ca' 'a'

Brute Force : *TC:$O(N^3)$, SC: $O(1)$*

- Iterate all the subarrays, and find out the substrings having count atmost k .

```
int countatmostk(string str, int k)
{
    int n = str.size();
    int res = 0;
 for(int i=0;i<n;i++)
 {
     for(int j=i; j<n; j++)
     {    int count=0;
           int m;
         vector<bool> visited(26);
         for(m=i; m<=j; m++)
         {
             if(visited[str[m]-'a']==true)
                  continue;
            visited[str[m]-'a']=true;
            count++;
         }
         if(count<=k)
         {
             res+=1;
         }

     }
 }
return res;
}
```

Better Approach - TC : $O(N^2)$, SC : $O(1)$

```cpp
int countatmostk(string str, int k) {

    int n = str.size();
    int res = 0;

 for(int i=0;i<n;i++)
 {  int j;
  vector<bool> visited(26);
  int count=0;
     for(j=i; j<n; j++)
     {
            if(visited[str[j]-'a']!=true)
            {  visited[str[j]-'a']=true;
            count++;
            }
    if(count<=k)
         {
             res+=1;
         }
     }

 }
return res;
    }
```

Note - Above program is only valid for lower case characters, if we want to program for unicode, we can use hashmap, instead of visited array.

```cpp
int countatmostk(string str, int k) {

    int n = str.size();
    int res = 0;

for(int i=0;i<n;i++)
{
   unordered_map<char,int> map;
    for(int j=i; j<n; j++)
    {    map[str[j]];
   int count = map.size();
   if(count<=k)
        {
            res+=1;
        }
    }
}
return res;
    }


int main() {
    string s = "aabcbcdbca";
    int k = 2;
   cout<<countatmostk(s, k);
    return 0;
}
//output :23
```

- Map will take O(256), so we can say. **SC =O(1)**

Optimal Approach : TC - O(N), SC - O(N)

Example: aabca, k=2

Character	Frequency
a	1 2 1 0
b	1 0
c	1
a	1

acquire	release	str[acquire]	dist_count	res	Map
0	0	a	1	1	Store 1 in map[a]
1	0	a	1	=1+2	map[a]++
2	0	b	2	=3+3=6	Store 1 in map[b]
3	0 1 2	c	3 3 2	=6+2=8	Store 1 in map[c] Map[a]--, map[a]--
4	2 3	a	3 2	=8+2=10	Store 1 in map[a] map[b]-

```cpp
int countatmostk(string str, int k)
{
    int dist_count=0;
    int res=0;
    int release=0;

    unordered_map<char,int> map;

    for(int acquire = 0; acquire < str.size(); acquire++)
    {    char curr_char = str[acquire];

       if(map.find(curr_char)≠map.end())
        map[curr_char]+=1;
       else
       {   dist_count+=1;
           map[curr_char]=1;
       }


        while((release≤acquire) && (dist_count>k))
        {
            char dis_char=str[release];
            map[dis_char]--;
            release++;

             if(map[dis_char]==0)
               { map.erase(dis_char);
                dist_count-=1;
               }

        }
        res+= (acquire-release+1);
 }
return res;
}
```

Without using distinct count variable :

```
int countatmostk(string str, int k)
{   int size=str.size();
    int res=0;
    int release=0;

    unordered_map<char,int> map;
    for(int acquire = 0; acquire < size; acquire++)
    {    int curr_char = str[acquire];
            map[curr_char]++;

        while(release<=acquire && map.size()>k)
        {
            int dis_char=str[release];
            map[dis_char]--;

             if(map[dis_char]==0)
               {
                    map.erase(dis_char);
               }
             release++;
        }
          res+=(acquire-release+1);
 }
return res;
}
```

Problem-19, HashMap

05 January 2022 10:36

Problem -19 - Largest Substring having atmost k unique characters

It is similar to problem 18,
Just put **res = -1**
And change line 45 as **res=max(res,(acquire-release+1));**

Example: aabca, k=2

Character	Frequency
a	1 2 1 0
b	1 0
c	1
a	1

acquire	release	str[acquire]	dist_count	res	Map
0	0	a	1	Max(-1,1)=1	Store 1 in map[a]
1	0	a	1	Max(1,2) =2	map[a]++
2	0	b	2	Max(2,3) =3	Store 1 in map[b]
3	0 1 2	c	3 3 2	Max(3,3)=3	Store 1 in map[c] Map[a]--, map[a]--
4	2 3	a	3 2	Max(3,2) = 3	Store 1 in map[a] map[b]--

```cpp
int countatmostk(string str, int k)
{   int size=str.size();
    int res=-1;
    int release=0;

    unordered_map<char,int> map;
    for(int acquire = 0; acquire < size; acquire++)
    {    int curr_char = str[acquire];
            map[curr_char]++;

        while(release<=acquire && map.size()>k)
        {
            int dis_char=str[release];
            map[dis_char]--;

             if(map[dis_char]==0)
               {
                    map.erase(dis_char);
               }
             release++;
        }
        //we dont need to check if(map.size()<=k)
        //program come here only if map.size<=k
        // if we need exactly k then we have to check the condition, because map size can be less than k
        res=max(res,(acquire-release+1));
 }
return res;
}
```

Problem-20, HashMap

06 January 2022 11:36

Problem-20: Count of substrings having atleast k unique characters

- Substrings having alteast K unique characters will be equal to
  T**otal substrings - substrings having atmost k-1 unique characters.**

```cpp
long long atMostKDistinctCharacters(string str, int k) {
        int size=str.size();
    int res=0;
    int release=0;

    unordered_map<char,int> map;
    for(int acquire = 0; acquire < size; acquire++)
    {    int curr_char = str[acquire];
        map[curr_char]++;

        while(release<=acquire && map.size()>k)
        {
            int dis_char=str[release];
            map[dis_char]--;

             if(map[dis_char]==0)
               {
                    map.erase(dis_char);
               }
             release++;
        }
          res+=(acquire-release+1);
 }
return res;
    }

    long long kDistinctCharacters(string &s, int k) {

        long long n = s.size();
        return (n * (n + 1) / 2) -
atMostKDistinctCharacters(s, k - 1);
    }
```

Problem-21, HashMap

06 January 2022 13:02

Problem-21: largest substring with atleast k unique characters

Its quite an very easy problem, we just need to check whether map contains atleast k distinct elements or not, as soon as we get atleast k unique elements, if we add anything in the string we will get atleast k uinque characters, so the largest substring will be nothing but the size of the string.

Example - 1 : s = {aaaaaaaaaaaaaaab}, k =3 , since there are only 2 distinct characters in the string, longest substring will be of length 0.
Example -2: s = {abcaaaaaaaaaaaaaaabbbbbbbccccccddddddddeeeeeeeeeeffffffffffff}, k=3, as soon as we will get three distinct elements, in the beginning itself we don't bother about the further characters, so the largest substring will be of same size of the size of given string.

```cpp
#include <iostream>
#include<vector>
#include<unordered_map>
using namespace std;

int countatleastk(string str, int k)
{   int size=str.size();
    int res=0;

    unordered_map<char,int> map;
    for(int acquire = 0; acquire < size; acquire++)
    {    int curr_char = str[acquire];
            map[curr_char]++;
        if(map.size()>=k)
         res=size;
 }
return res;
}

int main() {
    string s = "aaaaaaab";
    int k = 2;
   cout<<countatleastk(s, k);
    return 0;
}
```

Problem-22, HashMap

07 January 2022 22:25

Problem-21 : Two sum in a 2d Matrix

Problem Statement : Given a 2d matrix, we need to find a pair whose sum is equal to target, if there exists any such pair return true, else return false.

Example 1 :

Input:	Arr[][] ={12,13,14,15 16,18,17,31} Target = 30
Output :	True
Explaination:	{12,18} is such pair

Elements	Count
12	1
13	1
14	1
15	1
16	1
18	1
17	1
31	1

i=0, j=0, arr[i][j] = 12, remaining sum = 30-12 = 18,
18 is already present in the map, so return true.

```cpp
#include <iostream>
#include<unordered_map>
#include<vector>
using namespace std;

bool sum(vector<vector<int>>& v, int target)
{
    unordered_map<int,int> map;
    int rowsize = v.size();
    int colsize = v[0].size();

   for(int i=0;i<rowsize;i++)
   {
       for(int j=0;j<colsize;j++)
       {
           map[v[i][j]]++;
       }
}


for(int i=0;i<rowsize;i++)
   {
       for(int j=0;j<colsize;j++)
       {   int remainingsum = (target - v[i][j]);
```

```
    {
        for(int j=0;j<colsize;j++)
        {   int remainingsum = (target - v[i][j]);

            if(map.find(remainingsum)!=map.end())
            {
                 if((remainingsum==v[i][j]) && (map[remainingsum]>=2))

                      return true;
             return true;
            }

        }
}
return false;

}



int main()
 {
    int m=4;
    int n=2;
     int target;
   cin>>target;
    int key;
   vector<vector<int>> v;
   for(int i=0;i<m;i++)
   {   vector<int> v1;
       for(int j=0;j<n;j++)
       {    cin>>key;
           v1.push_back(key);
       }
       v.push_back(v1);
   }

 cout<<sum(v,target);
   return 0;
}

```

Edge case: if target = 30, and we got 15 in the array, we need 15 more to make pair, so if 15 exists in map its count should be 2.