Hashing Problems Kumar's Sheets

1.Count Frequencies of array

Given an array which may contain duplicates, print all elements and their frequencies.

Examples:

```
Input :  arr[] = {10, 20, 20, 10, 10, 20, 5, 20}
Output : 10 3
         20 4
         5  1

Input : arr[] = {10, 20, 20}
Output : 10 1
         20 2
```

A **simple solution** is to run two loops. For every item count number of times, it occurs. To avoid duplicate

```java
static void countFreq(int arr[], int n)
{
    Map<Integer, Integer> mp = new HashMap<>();

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++)
    {
        if (mp.containsKey(arr[i]))
        {
            mp.put(arr[i], mp.get(arr[i]) + 1);
        }
        else
        {
            mp.put(arr[i], 1);
        }
    }
    // Traverse through map and print frequencies
    for (Map.Entry<Integer, Integer> entry : mp.entrySet())
    {
        System.out.println(entry.getKey() + " " + entry.getValue());
    }
}
```

2.Two sum(Pair with given sum)

Input: arr[] = [0, -1, 2, -3, 1], target = -2
Output: true
Explanation: There is a pair (1, -3) with the sum equal to given target, 1 + (-3) = -2.

Input: arr[] = [1, -2, 1, 0, 5], target = 0
Output: false
Explanation: There is no pair with sum equals to given target.

```java
// User function Template for Java

class Solution {
    boolean twoSum(int arr[], int target) {
        // code here
        HashMap<Integer,Integer> map=new HashMap<>();
        for(int i=0;i<arr.length;i++){
            int b=target-arr[i];
            if(map.containsKey(b)){
                return true;
            }

            map.put(arr[i],i);
        }
        return false;
    }
}
```

3.Check if the array is subset of another array

Given two arrays **a[]** and **b[]** of size **m** and **n** respectively, the task is to determine whether **b[]** is a subset of **a[]**. Both arrays are not sorted, and elements are distinct.

Examples:

Input: *a[] = [11, 1, 13, 21, 3, 7], b[] = [11, 3, 7, 1]*
Output: *true*

Input: *a[]= [1, 2, 3, 4, 5, 6], b = [1, 2, 4]*
Output: *true*

Input: *a[] = [10, 5, 2, 23, 19], b = [19, 5, 3]*
Output: *false*

```java
//User function Template for Java

class Compute {
    public String isSubset( long a1[], long a2[], long n, long m) {


        Map<Long,Integer> map=new HashMap<>();

        for(long num : a1){
            map.put(num,map.getOrDefault(num,0)+1);
        }

        for(long num : a2){
            if(!map.containsKey(num) || map.get(num)==0){
                return "No";
            }

            map.put(num,map.get(num)-1);
        }
        return "Yes";


    }
```

4.Maximum distance between two occurrence

Given an array **arr[]**, the task is to find the **maximum distance** between two occurrences of any element.
If no element occurs twice, return 0.

Examples:

Input: *arr = [1, 1, 2, 2, 2, 1]*
Output: *5*
Explanation: *distance for 1 is: 5-0 = 5, distance for 2 is: 4-2 = 2, So max distance is 5.*

Input *: arr[] = [3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2]*
Output: *10*
Explanation : *Max distance for 2 is 11-1 = 10, max distance for 1 is 4-2 = 2 and max distance for 4 is 10-5 = 5*

Input: *arr[] = [1, 2, 3, 6, 5, 4]*
Output: *0*
Explanation: *No element has two occurrence, so maximum distance = 0.*

```java
class Solution {
    public int maxDistance(int[] arr) {
        // Code here
           HashMap<Integer, Integer> hm = new HashMap<>();
        int res = 0;

        for(int i = 0; i < arr.length; i++)
        {
            if(!hm.containsKey(arr[i]))
            {
                hm.put(arr[i], i);
            }
            else{
                res = Math.max(res, i - hm.get(arr[i]));
            }
        }
        return res;
    }
}
```

5.Minimum operation to make all elements equal in array

Given an array consisting of **n** positive integers, the task is to find the **minimum number** of operations to make all elements equal. In each operation, we can perform addition, multiplication, subtraction, or division with any number and an array element.

Examples:

Input *: arr[] = [1, 2, 3, 4]*
Output *: 3*
Explanation: *All element are different. Select any one element and make the other elements equal to selected element. For example, we can make them all 1 by doing three subtractions. Or make them all 3 by doing three additions.*

Input: *arr[] = [1, 2, 2, 3]*
Output: *2*
Explanation: *Perform operation on 1 and 3 to make them equal to 2.*

Input *: arr[] = [1, 1, 1, 1]*
Output *: 0*
Explanation: *All elements are equal.*

Step by step approach:

1. Create a hash map to store frequency counts for each unique value.
2. Populate the map by counting occurrences of each element in the array.
3. Find the maximum frequency among all elements in the map.
4. Calculate operations needed as (total elements – maximum frequency).

```java
// Function to find the minimum number of operations
static int minOperations(int[] arr) {
  int n = arr.length;

  // Store frequency of each element
  HashMap<Integer, Integer> freqMap = new HashMap<>();
  for (int i = 0; i < n; i++) {
    freqMap.put(arr[i], freqMap.getOrDefault(arr[i], 0) + 1);
  }

  // Find maximum frequency
  int maxFreq = 0;
  for (int val : freqMap.values()) {
    maxFreq = Math.max(maxFreq, val);
  }

  // Operations needed = total elements - maximum frequency
  return n - maxFreq;
}
```

6.Duplicate within k distance in an array

Given an integer array **arr[]** and an integer **k**, determine whether there exist two indices **i** and **j** such that **arr[i] == arr[j]** and **|i – j| ≤ k**. If such a pair exists, return **'Yes'**, otherwise return **'No'**.

Examples:

Input: *k = 3, arr[] = [1, 2, 3, 4, 1, 2, 3, 4]*
Output: *No*
Explanation: *Each element in the given array arr[] appears twice and the distance between every element and its duplicate is 4.*

Input: *k = 3, arr[] = [1, 2, 3, 1, 4, 5]*
Output: *Yes*
Explanation: *1 is present at index 0 and 3.*

Input: *k = 3, arr[] = [1, 2, 3, 4, 5]*
Output: *No*
Explanation: *There is no duplicate element in arr[].*

1. *Create an empty HashSet.*
2. *Traverse all elements from left to right. Let the current element be 'arr[i]'*
   - *If the current element 'arr[i]' is present in a HashSet, then return true.*
   - *Else add arr[i] to hash and remove arr[i-k] from hash if i >= k*

```java
   elements within k distance from each other */
import java.util.*;

class Main
{
    static boolean checkDuplicatesWithinK(int arr[], int k)
    {
        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array
        for (int i=0; i<arr.length; i++)
        {
            // If already present n hash, then we found
            // a duplicate within k distance
            if (set.contains(arr[i]))
               return true;

            // Add this item to hashset
            set.add(arr[i]);

            // Remove the k+1 distant item
            if (i >= k)
              set.remove(arr[i-k]);
        }
        return false;
    }
```

7.Sum of elements in an array with frequencies greater than or equal to that element

Sum of elements in an array with frequencies greater than or equal to that element

Last Updated : 19 Apr, 2023

Given an array arr[] of N integers. The task is to find the sum of the elements which have frequencies greater than or equal to that element in the array.

Examples:

```
Input: arr[] = {2, 1, 1, 2, 1, 6}
Output: 3
The elements in the array are {2, 1, 6}
Where,
 2 appear 2 times which is greater than equal to 2 itself.
 1 appear 3 times which is greater than 1 itself.
 But 6 appears 1 time which is not greater than or equals to 6.
So, sum = 2 + 1 = 3.

Input: arr[] = {1, 2, 3, 3, 2, 3, 2, 3, 3}
Output: 6
```

```java
// than or equal to that element
public static int sumOfElements(int[] arr, int n)
{

    // HashMap is used to calculate frequency of
    // elements of array
    HashMap<Integer, Integer> m
        = new HashMap<Integer, Integer>();
    for (int i = 0; i < n; i++) {
        if (m.containsKey(arr[i])) {
            m.put(arr[i], m.get(arr[i]) + 1);
        }
        else {
            m.put(arr[i], 1);
        }
    }

    int sum = 0;

    // Traverse the HashMap
    for (Integer key : m.keySet()) {

        // Calculate the sum of elements
        // having frequencies greater than
        // or equal to the element itself
        if (m.get(key) >= key) {
            sum += key;
        }
    }

    return sum;
}
```

8.First unique character in string

Example 1:

> **Input:** s = "leetcode"
>
> **Output:** 0
>
> **Explanation:**
>
> The character 'l' at index 0 is the first character that does not occur at any other inde

Example 2:

> **Input:** s = "loveleetcode"
>
> **Output:** 2

Example 3:

> **Input:** s = "aabb"
>
> **Output:** -1

```java
class Solution {
    public int firstUniqChar(String s) {

        HashMap<Character,Integer> map=new HashMap<>();

        for(char ch:s.toCharArray()){
            map.put(ch,map.getOrDefault(ch,0)+1);
        }

        for(int i=0;i<s.length();i++){
            if(map.get(s.charAt(i))==1){
                return i;
            }
        }
        return -1;
    }
}
```

9.Find Common Character

Given a string array `words`, return *an array of all characters that show up in all strings within the* `words` *(including duplicates)*. You may return the answer in **any order**.

Example 1:

> **Input:** words = ["bella","label","roller"]
> **Output:** ["e","l","l"]

Example 2:

> **Input:** words = ["cool","lock","cook"]
> **Output:** ["c","o"]

```java
public List<String> commonChars(String[] words) {
    Map<Character, Integer> commonMap = new HashMap<>();
    for (char c : words[0].toCharArray()) {
        commonMap.put(c, commonMap.getOrDefault(c, 0) + 1);
    }
    for (int i = 1; i < words.length; i++) {
        Map<Character, Integer> wordMap = new HashMap<>();
        for (char c : words[i].toCharArray()) {
            wordMap.put(c, wordMap.getOrDefault(c, 0) + 1);
        }
        for (char c : new HashSet<>(commonMap.keySet())) {
            if (wordMap.containsKey(c)) {
                commonMap.put(c, Math.min(commonMap.get(c), wordMap.get(c)));
            } else {
                commonMap.put(c, 0);
            }
        }
    }
    List<String> result = new ArrayList<>();
    for (Map.Entry<Character, Integer> entry : commonMap.entrySet()) {
        for (int i = 0; i < entry.getValue(); i++) {
            result.add(String.valueOf(entry.getKey()));
        }
    }

    return result;
}
```

10.2Sum-count pair with given sum

Given an array **arr[]** of **n** integers and a **target** value, the task is to find the number of pairs of integers in the array whose sum is equal to **target**.

Examples:

Input: *arr[] = {1, 5, 7, -1, 5}, target = 6*
Output: *3*
Explanation: *Pairs with sum 6 are (1, 5), (7, -1) & (1, 5).*

Input: *arr[] = {1, 1, 1, 1}, target = 2*
Output: *6*
Explanation: *Pairs with sum 2 are (1, 1), (1, 1), (1, 1), (1, 1), (1, 1) and (1, 1).*

Input: *arr[] = {10, 12, 10, 15, -1}, target = 125*
Output: *0*

```
class Solution {

    int countPairs(int arr[], int target) {
        // Your code here
        int count=0;
        HashMap<Integer,Integer> map=new HashMap<>();
        for(int i=0;i<arr.length;i++){
            int sec=target-arr[i];
            count += map.getOrDefault(sec, 0);


            map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);
        }
        return count;
    }
}
```

11.Count subarray having sum k

Given an unsorted array of integers, the task is to find the number of subarrays having a sum exactly equal to a given number **k**.

Examples:

Input : *arr[] = [10, 2, -2, -20, 10], k = -10*
Output : *3*
Explanation: *Subarrays: arr[0...3], arr[1...4], arr[3...4] have sum equal to -10.*

Input : *arr[] = [9, 4, 20, 3, 10, 5], k = 33*
Output : *2*
Explanation: *Subarrays: arr[0...2], arr[2...4] have sum equal to 33.*

Input : *arr[] = [1, 3, 5], k = 2*
Output : *0*
Explanation: *No subarrays with 0 sum.*

```java
// User function Template for Java
class Solution {
    public int countSubarrays(int arr[], int k) {
        // code here
        int sum=0;
        int count=0;

        HashMap<Integer,Integer> map=new HashMap<>();
        map.put(0,1);

        for(int num : arr){
            sum+=num;
            if(map.containsKey(sum-k)){
                count+=map.get(sum-k);
            }
            map.put(sum,map.getOrDefault(sum,0)+1);
        }
        return count;
    }
}
```

12.Smallest subarray with sum equal to k

I want to find the length smallest subarray whose sum is equal to k.

```
Input: arr[] = {2, 4, 6, 10, 2, 1}, K = 12
Output: 2
```

Explanation: All possible subarrays with sum 12 are `{2, 4, 6}` and `{10, 2}`.

```
Input: arr[] = { 1, 2, 4, 3, 2, 4, 1 }, K = 7
Output: 2
```

```java
class Solution {
    public int subarraySum(int[] nums, int k) {

        int count=0;
        int sum=0;

        HashMap<Integer,Integer> sumfreq=new HashMap<>();
        sumfreq.put(0,1);

        for(int num:nums){
            sum+=num;
            if(sumfreq.containsKey(sum-k)){
                count +=sumfreq.get(sum-k);
            }
            sumfreq.put(sum,sumfreq.getOrDefault(sum,0)+1);
        }
        return count;


    }
}
```

13.Subarray with given XOR

Given an array of integers **A** and an integer **B**.

Find the total number of subarrays having bitwise XOR of all elements equals to B.

Example Input

Input 1:

```
A = [4, 2, 2, 6, 4]
B = 6
```

Input 2:

```
A = [5, 6, 7, 8, 9]
B = 5
```

```java
public class Solution {
    public int solve(ArrayList<Integer> A, int B) {
        int n=A.size();
        int xor=0;

        HashMap<Integer,Integer> map=new HashMap<>();
        map.put(xor,1);
        int count=0;

        for(int i=0;i<n;i++){
            xor = xor^A.get(i);
            int x=xor^B;

            if(map.containsKey(x)){
                count +=map.get(x);
            }
            if(map.containsKey(xor)){
                map.put(xor,map.get(xor)+1);
            }else{
                map.put(xor,1);
            }
        }
        return count;
    }
}
```

14.Continous Subarray Sum

Example 1:

> **Input:** nums = [23,2,4,6,7], k = 6
> **Output:** true
> **Explanation:** [2, 4] is a continuous subarray of size 2 whose elements sum up to 6.

Example 2:

> **Input:** nums = [23,2,6,4,7], k = 6
> **Output:** true
> **Explanation:** [23, 2, 6, 4, 7] is an continuous subarray of size 5 whose elements sum up to 42.
> 42 is a multiple of 6 because 42 = 7 * 6 and 7 is an integer.

```java
class Solution {
    public boolean checkSubarraySum(int[] nums, int k) {
        HashMap<Integer,Integer> map=new HashMap<>();
        map.put(0,-1);

        int sum=0;
        for(int i=0;i<nums.length;i++){
            sum+=nums[i];
            int remainder=sum%k;

            //handle the negative values
            if(remainder<0){
                remainder +=k;
            }
            //check here to map atleast contains 2 element
            if(map.containsKey(remainder)){
                if(i-map.get(remainder)>1){
                    return true;
                }
            }else{
                map.put(remainder,i);
            }

        }
        return false;

    }
```

15.Subarray sum divisible by k

Given an integer array `nums` and an integer `k`, return *the number of non-empty **subarrays** that have a sum divisible by* `k`.

A **subarray** is a **contiguous** part of an array.

Example 1:

> **Input:** nums = [4,5,0,-2,-3,1], k = 5
> **Output:** 7
> **Explanation:** There are 7 subarrays with a sum divisible by k = 5:
> [4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]

Example 2:

> **Input:** nums = [5], k = 9
> **Output:** 0

```java
class Solution {
    public int subarraysDivByK(int[] nums, int k) {
        HashMap<Integer,Integer> map=new HashMap<>();
        map.put(0,1);

        int sum=0;
        int res=0;
        for(int i=0;i<nums.length;i++){
            sum +=nums[i];
            int mod=sum%k;
            //convert negative sum to +ve
            if(mod<0){
                mod=mod%k+k;
            }
            if(map.containsKey(mod)){
                res +=map.get(mod);
            }

            map.put(mod,map.getOrDefault(mod,0)+1);
        }
        return res;

    }
}
```

16.Max number of k-sum of pairs

You are given an integer array `nums` and an integer `k`.

In one operation, you can pick two numbers from the array whose sum equals `k` and remove them from the array.

Return *the maximum number of operations you can perform on the array.*

Example 1:

```
Input: nums = [1,2,3,4], k = 5
Output: 2
Explanation: Starting with nums = [1,2,3,4]:
- Remove numbers 1 and 4, then nums = [2,3]
- Remove numbers 2 and 3, then nums = []
There are no more pairs that sum up to 5, hence a total of 2 operations.
```

Example 2:

```
Input: nums = [3,1,3,4,3], k = 6
Output: 1
Explanation: Starting with nums = [3,1,3,4,3]:
- Remove the first two 3's, then nums = [1,4,3]
There are no more pairs that sum up to 6, hence a total of 1 operation.
```

```
;s Solution {
 public int maxOperations(int[] nums, int k) {
    HashMap<Integer,Integer>map=new HashMap<>();
    int count=0;
    for(int i=0;i<nums.length;i++){
        //to check if that k-nums[i] present and had some value left or already paired
        if(map.containsKey(k-nums[i])&&map.get(k-nums[i])>0){
            count++;
            map.put(k-nums[i],map.get(k-nums[i])-1);
        }else{
            //getOrDefault is easy way it directly checks if value is 0 returns 0 where I a
            //and if some value is present then it return that value "similar to map.get(i)
            map.put(nums[i],map.getOrDefault(nums[i],0)+1);
        }
    }
    return count;
}
```

```
//optimised--O(logn)
       int count=0;    ------→Using Two pointer
       int n=nums.length;
       Arrays.sort(nums);
       int left=0;
       int right=n-1;
       while(left<right){
           if(nums[left]+nums[right]<k){
               left++;
           }else if(nums[left]+nums[right]>k){
               right--;
           }else{
               count++;
               left++;
               right--;
           }
       }
       return count;
```

17.Count Number of pairs with absolute difference k

Given an integer array `nums` and an integer `k`, return *the number of pairs* `(i, j)` *where* `i < j` *such that* `|nums[i] - nums[j]| == k`.

The value of `|x|` is defined as:

- `x` if `x >= 0`.
- `-x` if `x < 0`.

Example 1:

Input: nums = [1,2,2,1], k = 1
Output: 4
Explanation: The pairs with an absolute difference of 1 are:
- [**1**,**2**,2,1]
- [**1**,2,**2**,1]
- [1,**2**,2,**1**]
- [1,2,**2**,**1**]

Example 2:

Input: nums = [1,3], k = 3
Output: 0
Explanation: There are no pairs with an absolute difference of 3.

```java
class Solution {
    public int countKDifference(int[] nums, int k) {
        Map<Integer,Integer> map = new HashMap<>();
        int res = 0;

        for(int i = 0;i< nums.length;i++){
            if(map.containsKey(nums[i]-k)){
                res+= map.get(nums[i]-k);
            }
            if(map.containsKey(nums[i]+k)){
                res+= map.get(nums[i]+k);
            }
            map.put(nums[i],map.getOrDefault(nums[i],0)+1);
        }


        return res;
    }
}
```

18.Number of pair of string with concatenation equals to target.

Given an array of **digit** strings `nums` and a **digit** string `target`, return *the number of pairs of indices* `(i, j)` *(where* `i != j`*) such that the* ***concatenation*** *of* `nums[i] + nums[j]` *equals* `target`.

Example 1:

Input: nums = ["777","7","77","77"], target = "7777"
Output: 4
Explanation: Valid pairs are:
- (0, 1): "777" + "7"
- (1, 0): "7" + "777"
- (2, 3): "77" + "77"
- (3, 2): "77" + "77"

Example 2:

Input: nums = ["123","4","12","34"], target = "1234"
Output: 2
Explanation: Valid pairs are:
- (0, 1): "123" + "4"
- (2, 3): "12" + "34"

```java
class Solution {
    public int numOfPairs(String[] nums, String target) {

        HashMap<String,Integer> map=new HashMap<>();

        int count=0;

        for(int i=0;i<nums.length;i++){
            if(target.startsWith(nums[i])){
                String sub=target.substring(nums[i].length());
                if(map.containsKey(sub)){
                    count+=map.get(sub);
                }
            }
            if(target.endsWith(nums[i])){
                String sub=target.substring(0,target.length()-nums[i].length());
                if(map.containsKey(sub)){
                    count+=map.get(sub);
                }
            }
            map.put(nums[i],map.getOrDefault(nums[i],0)+1);
        }
        return count;
    }
}
```

19. 3-Sum Triplet sum in array

Given an array **arr[]** of size **n** and an integer **sum**, the task is to check if there is a triplet in the array which sums up to the given **target** sum.

Examples:

Input: *arr[] = [1, 4, 45, 6, 10, 8], target = 13*
Output: *true*
Explanation: *The triplet [1, 4, 8] sums up to 13*

Input: *arr[] = [1, 2, 4, 3, 6, 7], target = 10*
Output: *true*
Explanation: *The triplets [1, 3, 6] and [1, 2, 7] both sum to 10.*

Input: *arr[] = [40, 20, 10, 3, 6, 7], sum = 24*
Output: *false*
Explanation: *No triplet in the array sums to 24.*

```java
// User function Template for Java

class Solution {
    // Should return true if there is a triplet with sum equal
    // to x in arr[], otherwise false
    public static boolean hasTripletSum(int arr[], int target) {
        // Your code Here
        int n=arr.length;
        for(int i=0;i<n-2;i++){
            HashSet<Integer> set=new HashSet<>();
            for(int j=i+1;j<n;j++){
                int second=target-arr[i]-arr[j];
                if(set.contains(second)){
                    return true;
                }
                set.add(arr[j]);
            }
        }
        return false;
    }
}
```

20. 4 Sum –Count quadruplets with given sum in an array

Given an array **arr[]** and a **target** value, the task is to find the count of quadruplets present in the given array having sum equal to the given target.

Examples:

Input: *arr[] = [1, 5, 3, 1, 2, 10], target = 20*
Output: *1*
Explanation: *Only quadruplet satisfying the conditions is arr[1] + arr[2] + arr[4] + arr[5] = 5 + 3 + 2 + 10 = 20.*

Input: *arr[] = [1, 1, 1, 1, 1], target = 4*
Output: *5*
Explanation:
arr[0] + arr[1] + arr[2] + arr[3] = 4
arr[0] + arr[1] + arr[3] + arr[4] = 4
arr[0] + arr[1] + arr[2] + arr[4] = 4
arr[0] + arr[2] + arr[3] + arr[4] = 4
arr[1] + arr[2] + arr[3] + arr[4] = 4

Input: *arr = [4, 3, -13, 3], target = -3*
Output: *1*
Explanation: *There is only 1 quadruplet with sum = -3, that is [4, 3, -13, 3].*

```java
class Solution {
    public int countSum(int arr[], int target) {
        // code here
        int n=arr.length;
        int count=0;
        HashMap<Integer,Integer> map=new HashMap<>();
        for(int i=0;i<n-1;i++){
            for(int j=i+1;j<n;j++){
                int temp=arr[i]+arr[j];
                count+=map.getOrDefault(target-temp,0);
            }
            for(int j=0;j<i;j++){
                int temp=arr[i]+arr[j];
                map.put(temp,map.getOrDefault(temp,0)+1);

            }
        }
        return count;

    }
}
```

21.Maximum Sum of distinct subarrays with length k

You are given an integer array `nums` and an integer `k`. Find the maximum subarray sum of all the subarrays of `nums` that meet the following conditions:

- The length of the subarray is `k`, and
- All the elements of the subarray are **distinct**.

Return *the maximum subarray sum of all the subarrays that meet the conditions.* If no subarray meets the conditions, return `0`.

*A **subarray** is a contiguous non-empty sequence of elements within an array.*

Example 1:

```
Input: nums = [1,5,4,2,9,9,9], k = 3
Output: 15
Explanation: The subarrays of nums with length 3 are:
- [1,5,4] which meets the requirements and has a sum of 10.
- [5,4,2] which meets the requirements and has a sum of 11.
- [4,2,9] which meets the requirements and has a sum of 15.
- [2,9,9] which does not meet the requirements because the element 9 is
repeated.
- [9,9,9] which does not meet the requirements because the element 9 is
repeated.
We return 15 because it is the maximum subarray sum of all the subarrays
that meet the conditions
```

Example 2:

```
Input: nums = [4,4,4], k = 3
Output: 0
Explanation: The subarrays of nums with length 3 are:
- [4,4,4] which does not meet the requirements because the element 4 is
repeated.
We return 0 because no subarrays meet the conditions.
```

```java
class Solution {
    public long maximumSubarraySum(int[] nums, int k) {
        Set<Integer> set = new HashSet<>();
        long max = 0, sum = 0;
        int windowStart = 0;
        for (int i = 0; i < nums.length; i++) {
            while (set.contains(nums[i]) || set.size() == k) {
                set.remove(nums[windowStart]);
                sum -= nums[windowStart++];
            }
            sum += nums[i];
            set.add(nums[i]);
            if (set.size() == k) {
                max = Math.max(max, sum);
            }
        }
        return max;
    }
}
```

22.Count array pairs divisible by k

Given a **0-indexed** integer array `nums` of length `n` and an integer `k`, return *the* ***number of pairs*** `(i, j)` *such that:*

- `0 <= i < j <= n - 1` *and*
- `nums[i] * nums[j]` *is divisible by* `k`.

Example 1:

Input: nums = [1,2,3,4,5], k = 2
Output: 7
Explanation:
The 7 pairs of indices whose corresponding products are divisible by 2 are
(0, 1), (0, 3), (1, 2), (1, 3), (1, 4), (2, 3), and (3, 4).
Their products are 2, 4, 6, 8, 10, 12, and 20 respectively.
Other pairs such as (0, 2) and (2, 4) have products 3 and 15 respectively, which are not divisible by 2.

```java
class Solution {
    public long countPairs(int[] nums, int k) {
        long result = 0;
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int gcd = findGcd(nums[i], k);
            for (int num : map.keySet()) {
                if ((long) gcd * num % k == 0) { // Need to c
                    result += map.get(num);
                }
            }
            map.put(gcd, map.getOrDefault(gcd, 0) + 1);
        }
        return result;
    }
    private int findGcd(int x, int y) {
        if (x < y) {
            return findGcd(y, x);
        }
        return y == 0 ? x : findGcd(y, x % y);
    }
}
```

Important hashing pdf for revision

https://drive.google.com/file/d/1SVAHLw6pGg1wfcxflP8P-YmuT9FC-uWM/view