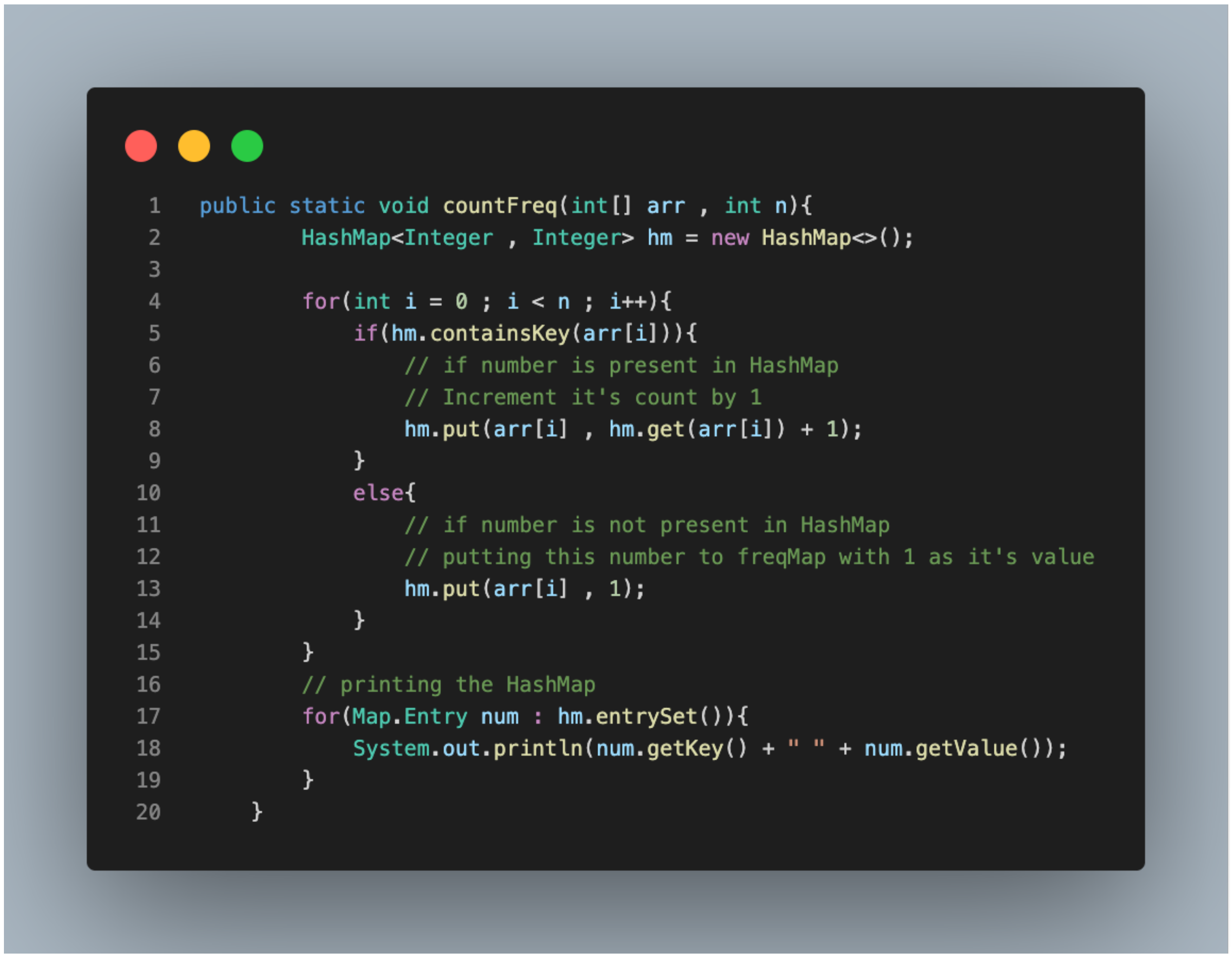
SET - A

P0. Counting Frequencies of array elements.

Input : arr[] = {10 , 20 , 20 , 10 , 10 , 20 , 5 , 20}

Output : 10   3
20 4
5   1

Logic : Time : O(N) (using single loop to track hashmap)
Space : O(N) (due to hashmap)



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

        for(int i = 0 ; i < n ; i++){
            if(hm.containsKey(arr[i])){
                // if number is present in HashMap
                // Increment it's count by 1
                hm.put(arr[i] , hm.get(arr[i]) + 1);
            }
            else{
                // if number is not present in HashMap
                // putting this number to freqMap with 1 as it's value
                hm.put(arr[i] , 1);
            }
        }
        // printing the HashMap
        for(Map.Entry num : hm.entrySet()){
            System.out.println(num.getKey() + " " + num.getValue());
        }
    }
```

P1. Check if a pair exists with given sum in given array (TWO SUM)

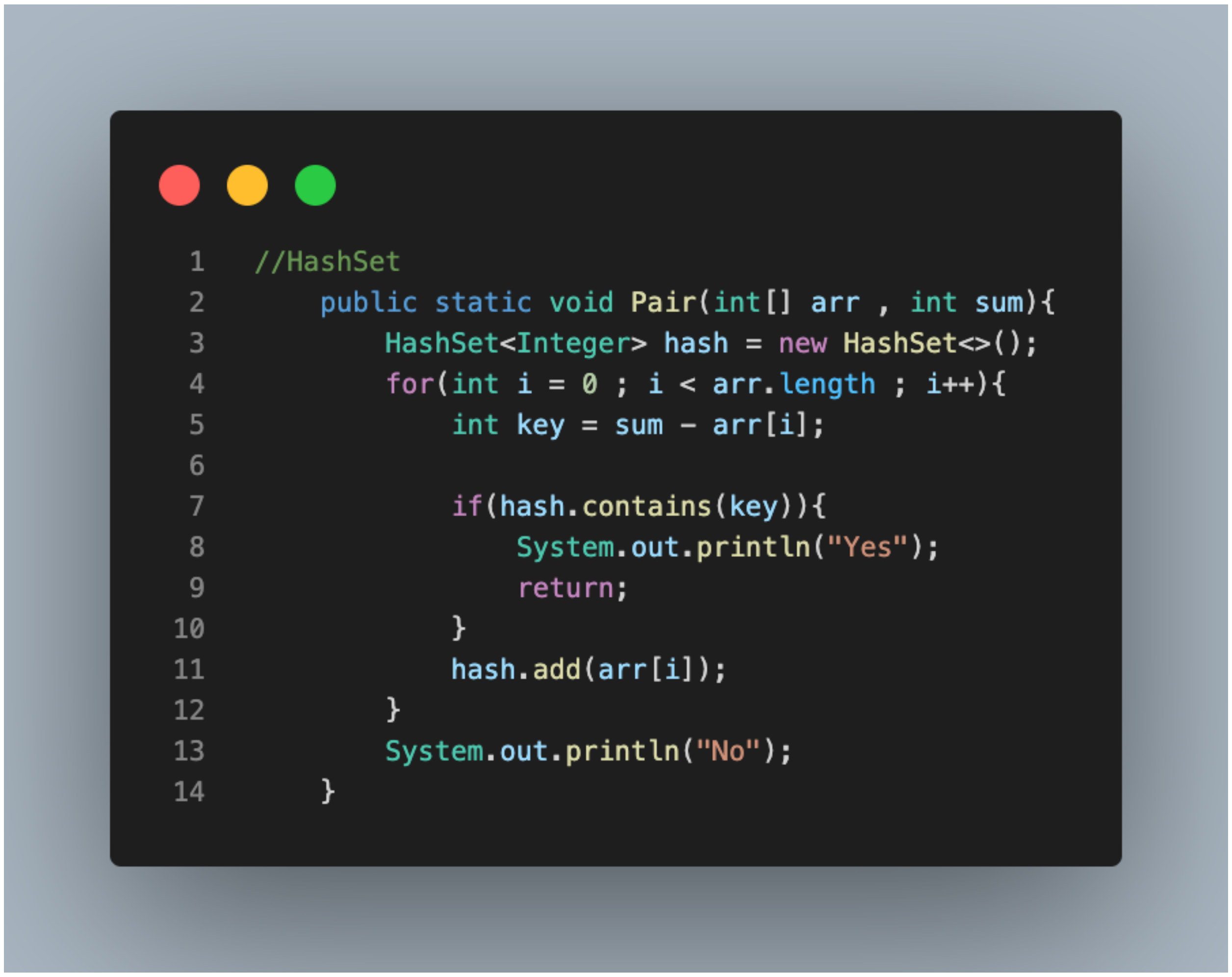
```
input: arr[] = {0, -1, 2, -3, 1}, x= -2
Output: Yes
Explanation:  If we calculate the sum of the output,1 + (-3) = -2
```

Logic :

```
arr[] = {0, -1, 2, -3, 1}
```

```
sum = -2
key = sum - arr[i]
Now start traversing:
Step 1: For '0' there is no valid number '-2' so store '0' in hash_map.
Step 2: For '-1' there is no valid number '-1' so store '-1' in hash_map.
Step 3: For '2' there is no valid number '-4' so store '2' in hash_map.
Step 4: For '-3' there is no valid number '1' so store '-3' in hash_map.
Step 5: For '1' there is a valid number '-3' so answer is 1, -3
```

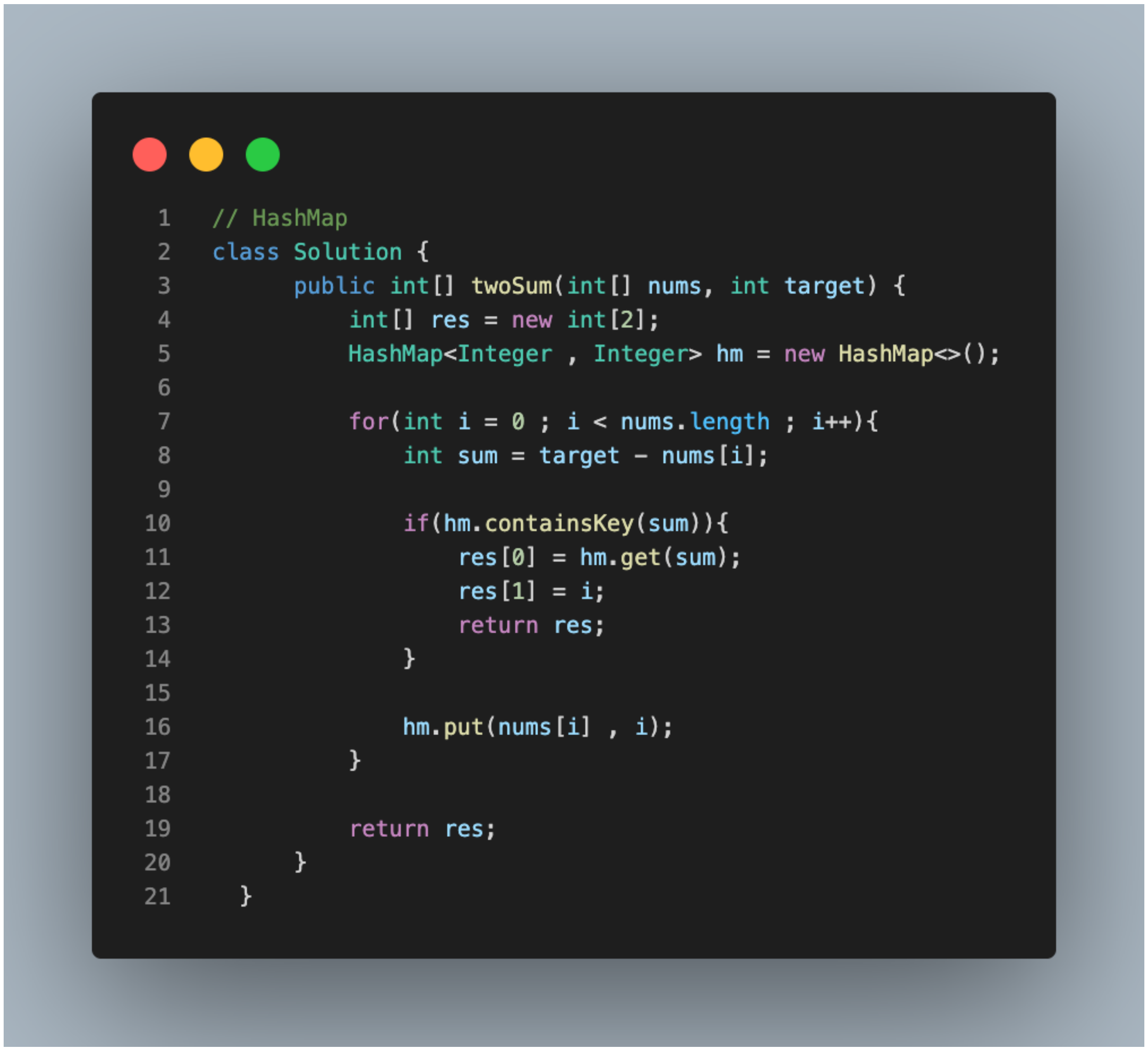
CODE :
HashSet



```java
//HashSet
    public static void Pair(int[] arr , int sum){
        HashSet<Integer> hash = new HashSet<>();
        for(int i = 0 ; i < arr.length ; i++){
            int key = sum - arr[i];

            if(hash.contains(key)){
                System.out.println("Yes");
                return;
            }
            hash.add(arr[i]);
        }
        System.out.println("No");
    }
```

HashMap(Leetcode : Two Sum)

We can solve this problem efficiently by using Hashing. We'll use a HashMap to see if there exists a value target – x for each value x. If target – x is found in the map, can return current element x's index and (target-x)'s index



```java
// HashMap
class Solution {
        public int[] twoSum(int[] nums, int target) {
            int[] res = new int[2];
            HashMap<Integer , Integer> hm = new HashMap<>();

            for(int i = 0 ; i < nums.length ; i++){
                int sum = target - nums[i];

                if(hm.containsKey(sum)){
                    res[0] = hm.get(sum);
                    res[1] = i;
                    return res;
                }

                hm.put(nums[i] , i);
            }

            return res;
        }
    }
```

P2. Find whether an array is subset of another array

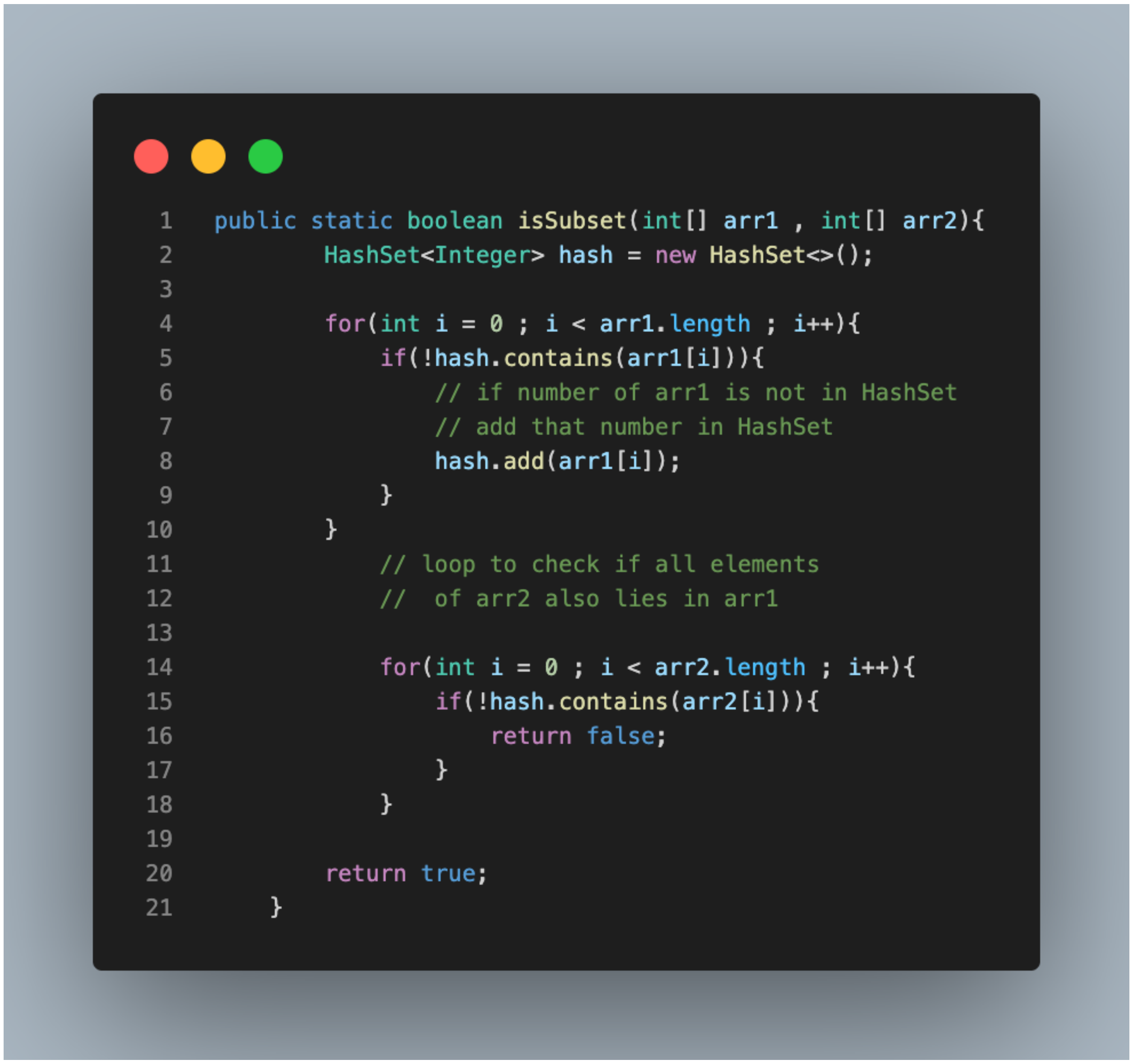
```
Input: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}
Output: arr2[] is a subset of arr1[]
```

Approach 1 :

```
Given array arr1[] = { 11, 1, 13, 21, 3, 7 } and arr2[] = { 11, 3, 7, 1 }.

Step 1: We will store the array arr1[] elements in HashSet

Step 2: We will look for each element in arr2[] in arr1[] using binary search.

arr2[] = { 11, 3, 7, 1 }, 11 is present in the HashSet = { 1, 3, 7, 11, 13, 21}
arr2[] = { 11, 3, 7, 1 }, 3 is present in the HashSet = { 1, 3, 7, 11, 13, 21}
arr2[] = { 11, 3, 7, 1 }, 7 is present in the HashSet = { 1, 3, 7, 11, 13, 21}
```

```
arr2[] = { 11, 3, 7, 1 }, 1 is present in the HashSet = { 1, 3, 7, 11, 13, 21}
As all the elements are found we can conclude arr2[] is the subset of arr1[].
```

HashSet : Time : O(NlogN)   , Space : O(N)



```java
public static boolean isSubset(int[] arr1 , int[] arr2){
        HashSet<Integer> hash = new HashSet<>();

        for(int i = 0 ; i < arr1.length ; i++){
            if(!hash.contains(arr1[i])){
                // if number of arr1 is not in HashSet
                // add that number in HashSet
                hash.add(arr1[i]);
            }
        }
            // loop to check if all elements
            //  of arr2 also lies in arr1

            for(int i = 0 ; i < arr2.length ; i++){
                if(!hash.contains(arr2[i])){
                    return false;
                }
            }

        return true;
    }
```

Approach 2 :

```
Given array arr1[] = { 11, 1, 13, 21, 3, 7 } and arr2[] = { 11, 3, 7, 1 }.


Step 1: We will store the array arr1[] and arr2[] elements in Set


The final Set = { 1, 3, 7, 11, 13, 21}
Step 2: Size of arr1[] = 6 and size of the Set = 6


Hence no new elements are found in arr2[]
As all the elements are found we can conclude arr2[] is the subset of arr1[].
```

HashSet : Time : O(m+n)

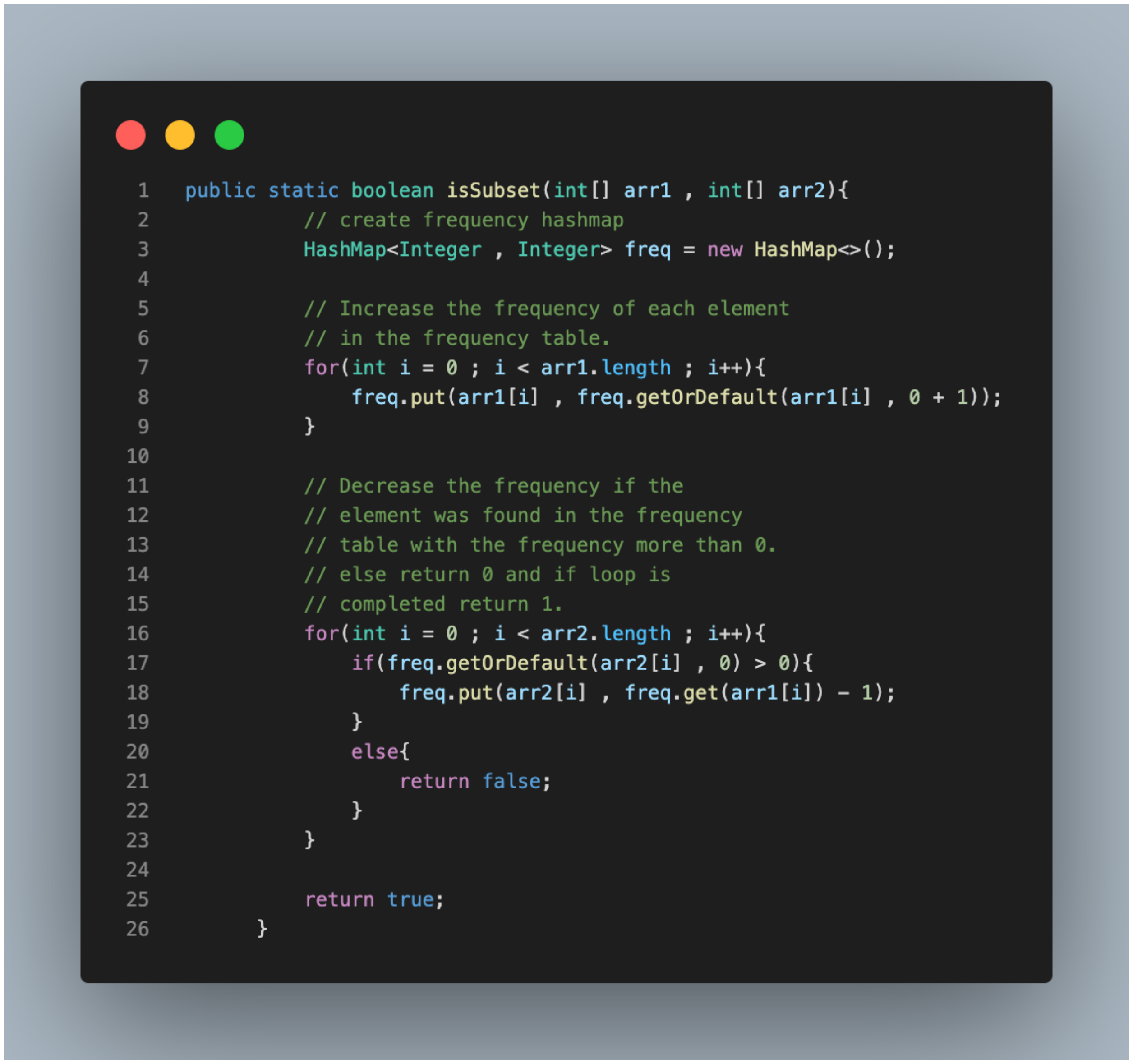
Space : O(n + m)

```
public static boolean isSubset(int[] arr1 , int[] arr2){
       HashSet<Integer> hash = new HashSet<>();

       for(int i = 0 ; i < arr1.length ; i++){
           hash.add(arr1[i]);
       }

       int hash_size = hash.size();

       for(int i = 0 ; i < arr2.length ; i++){
           hash.add(arr2[i]);
       }

       if(hash.size() == hash_size){
           return true;
       }
       else{
           return false;
       }
   }
```

Approach 3 :

```
Given array arr1[] = { 11, 1, 13, 21, 3, 7 } and arr2[] = { 11, 3, 7, 1 }.

Step 1: We will store the array arr1[] elements frequency in the frequency array

The frequency array will look like this

Step 2: We will look for arr2[] elements in the frequency array.

arr2[] = { 11, 3, 7, 1 }, 11 is present in the frequency array
arr2[] = { 11, 3, 7, 1 }, 3 is present in the frequency array
arr2[] = { 11, 3, 7, 1 }, 7 is present in the frequency array
arr2[] = { 11, 3, 7, 1 }, 1 is present in the frequency array
As all the elements are found we can conclude arr2[] is the subset of arr1[].
```

HashMap : Time : O(m + n)
Space : O(n)



```java
public static boolean isSubset(int[] arr1 , int[] arr2){
          // create frequency hashmap
          HashMap<Integer , Integer> freq = new HashMap<>();

          // Increase the frequency of each element
          // in the frequency table.
          for(int i = 0 ; i < arr1.length ; i++){
              freq.put(arr1[i] , freq.getOrDefault(arr1[i] , 0 + 1));
          }

          // Decrease the frequency if the
          // element was found in the frequency
          // table with the frequency more than 0.
          // else return 0 and if loop is
          // completed return 1.
          for(int i = 0 ; i < arr2.length ; i++){
              if(freq.getOrDefault(arr2[i] , 0) > 0){
                  freq.put(arr2[i] , freq.get(arr1[i]) - 1);
              }
              else{
                  return false;
              }
          }

          return true;
      }
```

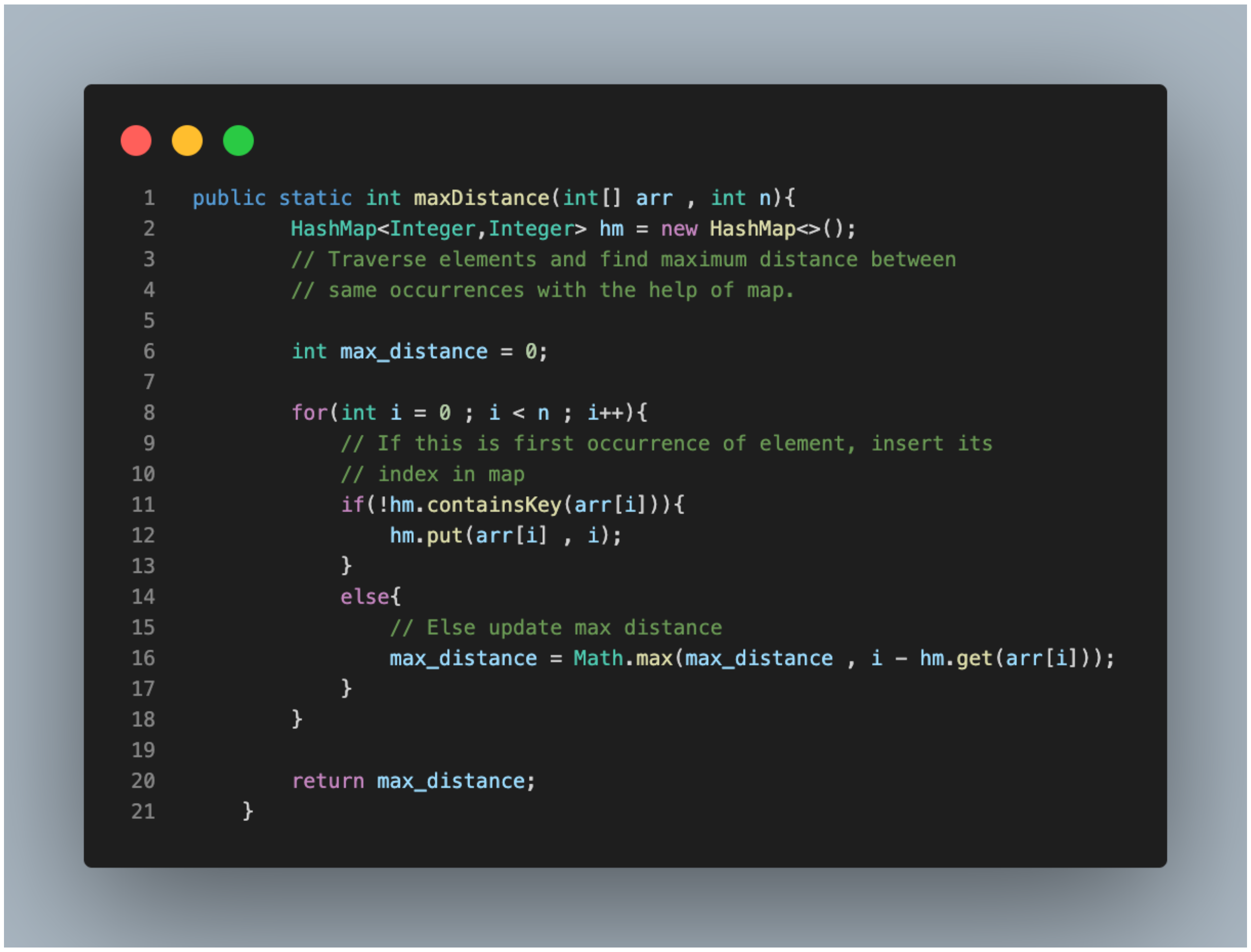
P3. Maximum distance between two occurrences of same element in array

```
Input : arr[] = {3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2}
Output: 10
maximum distance for 2 is 11-1 = 10
maximum distance for 1 is 4-2 = 2
maximum distance for 4 is 10-5 = 5
```

Approach :

```
An efficient solution to this problem is to use hashing. The idea is to traverse the
input array and store the index of the first occurrence in a hash map. For every other
occurrence, find the difference between the index and the first index stored in the hash map.
If the difference is more than the result so far, then update the result.
```

HashMap : Time : O(N)
Space : O(N)



```java
public static int maxDistance(int[] arr , int n){
        HashMap<Integer,Integer> hm = new HashMap<>();
        // Traverse elements and find maximum distance between
        // same occurrences with the help of map.

        int max_distance = 0;

        for(int i = 0 ; i < n ; i++){
            // If this is first occurrence of element, insert its
            // index in map
            if(!hm.containsKey(arr[i])){
                hm.put(arr[i] , i);
            }
            else{
                // Else update max distance
                max_distance = Math.max(max_distance , i - hm.get(arr[i]));
            }
        }

        return max_distance;
    }
```

P4. Minimum operation to make all elements equal in array

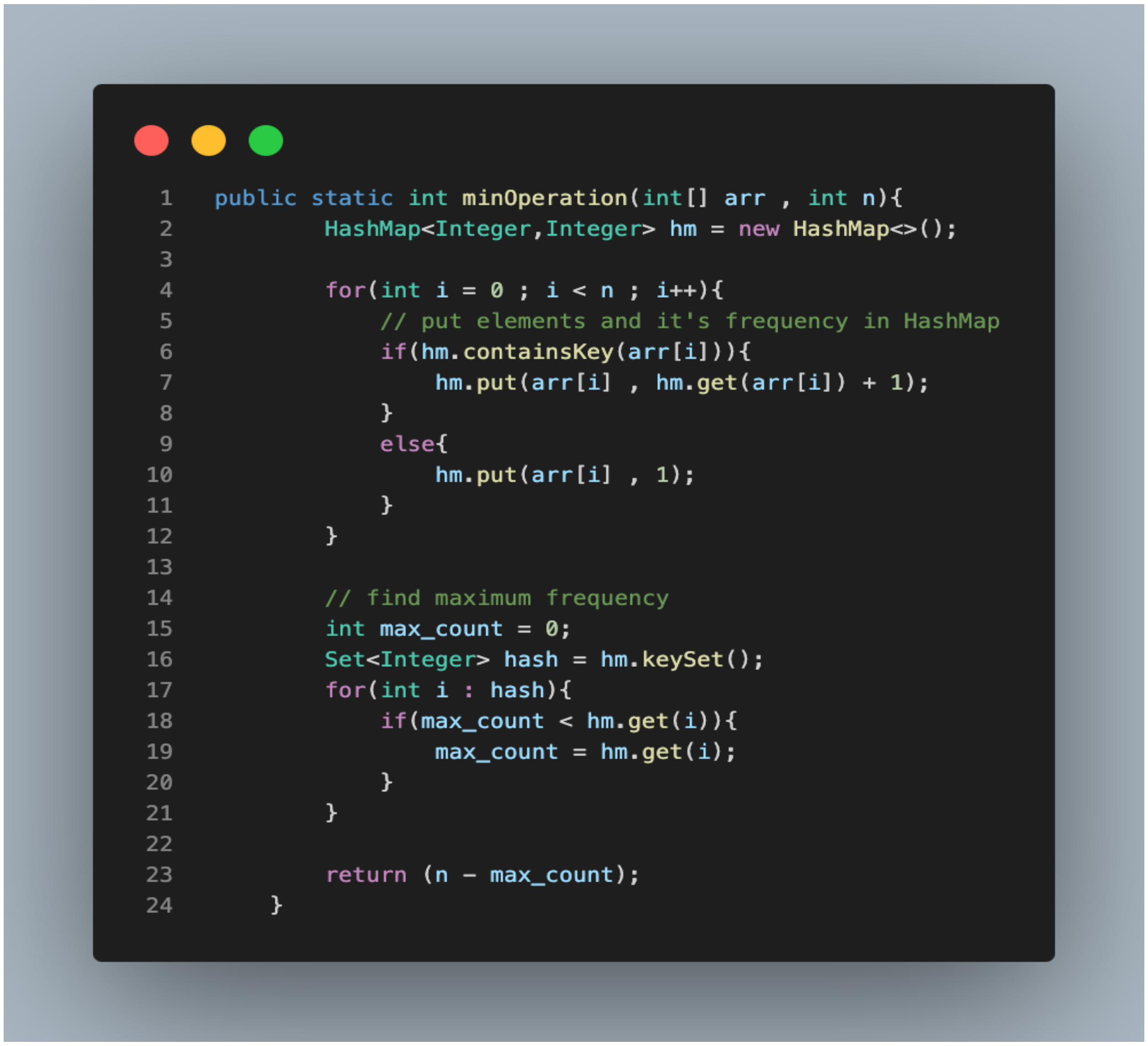
```
Input : arr[] = {1, 2, 3, 4}
Output : 3
Since all elements are different,
we need to perform at least three
operations to make them same. For
example, we can make them all 1
by doing three subtractions. Or make
them all 3 by doing three additions.
```

Approach :

```
For making all elements equal you can select a target value and then you can make all elements equal to
that.
Now, for converting a single element to target value you can perform a single operation only once. In
this manner
```

```
you can achieve your task in maximum of n operations but you have to minimize this number of operation
and for this
your selection of target is very important because if you select a target whose frequency in array is x
then you have
to perform only n-x more operations as you have already x elements equal to your target value. So
finally, our task is
reduced to finding the element with maximum frequency. This can be achieved by different means such as
iterative method
in O(n^2), sorting in O(nlogn) and hashing in O(n) time complexity.
```

HashMap : Time : O(N)
Space : O(N)



```java
public static int minOperation(int[] arr , int n){
        HashMap<Integer,Integer> hm = new HashMap<>();

        for(int i = 0 ; i < n ; i++){
            // put elements and it's frequency in HashMap
            if(hm.containsKey(arr[i])){
                hm.put(arr[i] , hm.get(arr[i]) + 1);
            }
            else{
                hm.put(arr[i] , 1);
            }
        }

        // find maximum frequency
        int max_count = 0;
        Set<Integer> hash = hm.keySet();
        for(int i : hash){
            if(max_count < hm.get(i)){
                max_count = hm.get(i);
            }
        }

        return (n - max_count);
    }
```

P5. Count maximum points on same line (Doubt)

```
Input : points[] = {-1, 1}, {0, 0}, {1, 1},
                   {2, 2}, {3, 3}, {3, 4}
```

```
Output : 4
Then maximum number of point which lie on same
line are 4, those point are {0, 0}, {1, 1}, {2, 2},
{3, 3}
```

Approach :

```
We can solve above problem by following approach – For each point p, calculate its slope with other
points
and use a map to record how many points have same slope, by which we can find out how many points are
on same
line with p as their one point. For each point keep doing the same thing and update the maximum number
of point
count found so far.

Some things to note in implementation are:

if two point are (x1, y1) and (x2, y2) then their slope will be (y2 – y1) / (x2 – x1) which can be a
double value
and can cause precision problems. To get rid of the precision problems, we treat slope as pair ((y2 –
y1), (x2 – x1))
instead of ratio and reduce pair by their gcd before inserting into map. In below code points which are
vertical or
repeated are treated separately.
```

HashMap : Time : O(N^2 log N)
Space : O(N)

P6. Check if a given array contains duplicate elements within k distance from each other

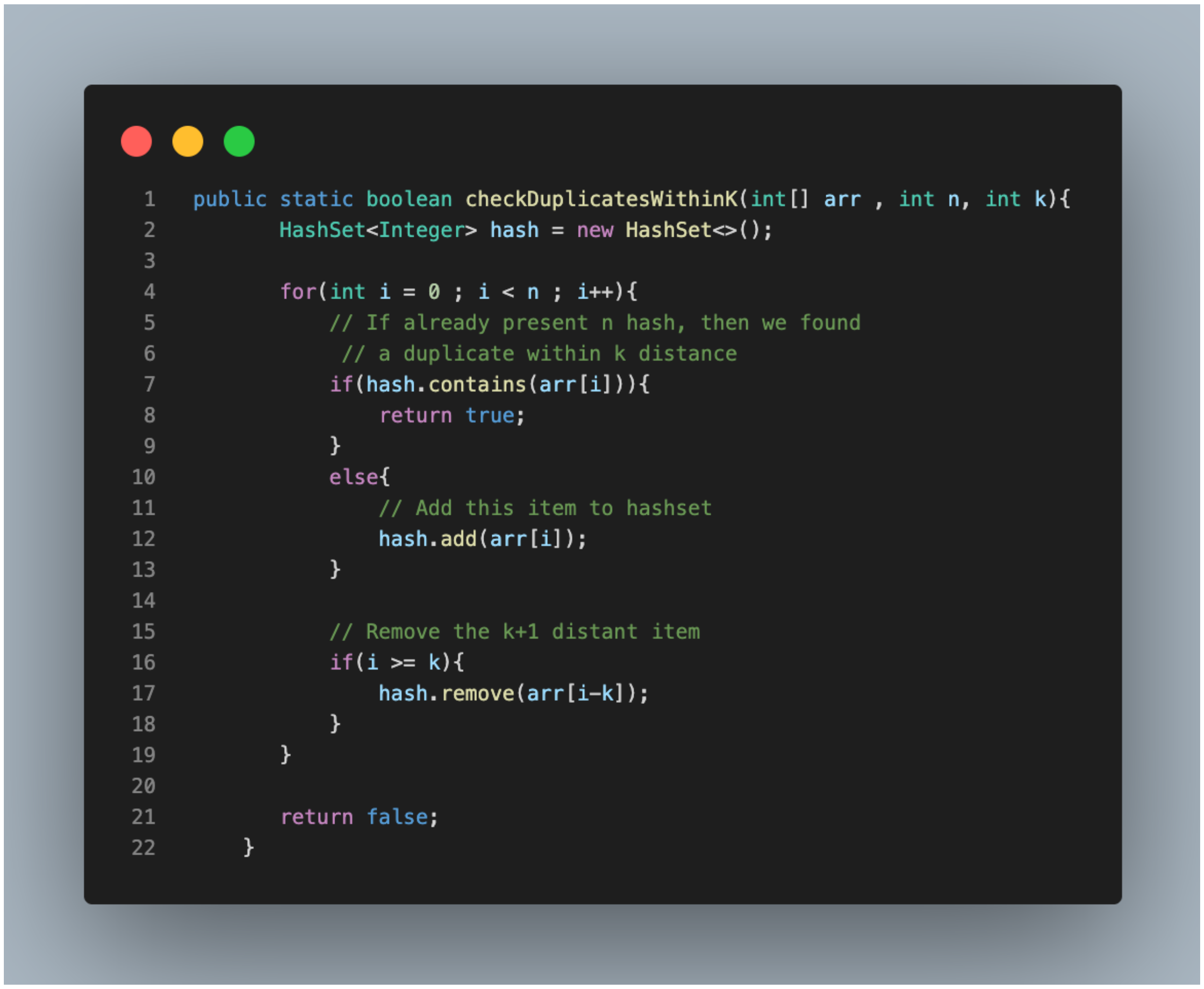
```
Input: k = 3, arr[] = {1, 2, 3, 4, 1, 2, 3, 4}
Output: false
All duplicates are more than k distance away.

Input: k = 3, arr[] = {1, 2, 3, 1, 4, 5}
Output: true
1 is repeated at distance 3.

Input: k = 3, arr[] = {1, 2, 3, 4, 5}
Output: false

Input: k = 3, arr[] = {1, 2, 3, 4, 4}
Output: true
```

Approach :

```
We can solve this problem in Θ(n) time using Hashing. The idea is to add elements to the hash.
We also remove elements that are at more than k distance from the current element. Following is a
detailed algorithm.

1. Create an empty hashtable.
```

```
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 hashtable, then return true.
-> Else add arr[i] to hash and remove arr[i-k] from hash if i is greater than or equal to k
```

HashMap : Time : O(N)
Space : O(N)



```java
public static boolean checkDuplicatesWithinK(int[] arr , int n, int k){
        HashSet<Integer> hash = new HashSet<>();

        for(int i = 0 ; i < n ; i++){
            // If already present n hash, then we found
             // a duplicate within k distance
            if(hash.contains(arr[i])){
                return true;
            }
            else{
                // Add this item to hashset
                hash.add(arr[i]);
            }

            // Remove the k+1 distant item
            if(i >= k){
                hash.remove(arr[i-k]);
            }
        }

        return false;
    }
```

P7. Sum of elements in an array with frequencies greater than or equal to that element (DOUBT)

```
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
```

Approach :

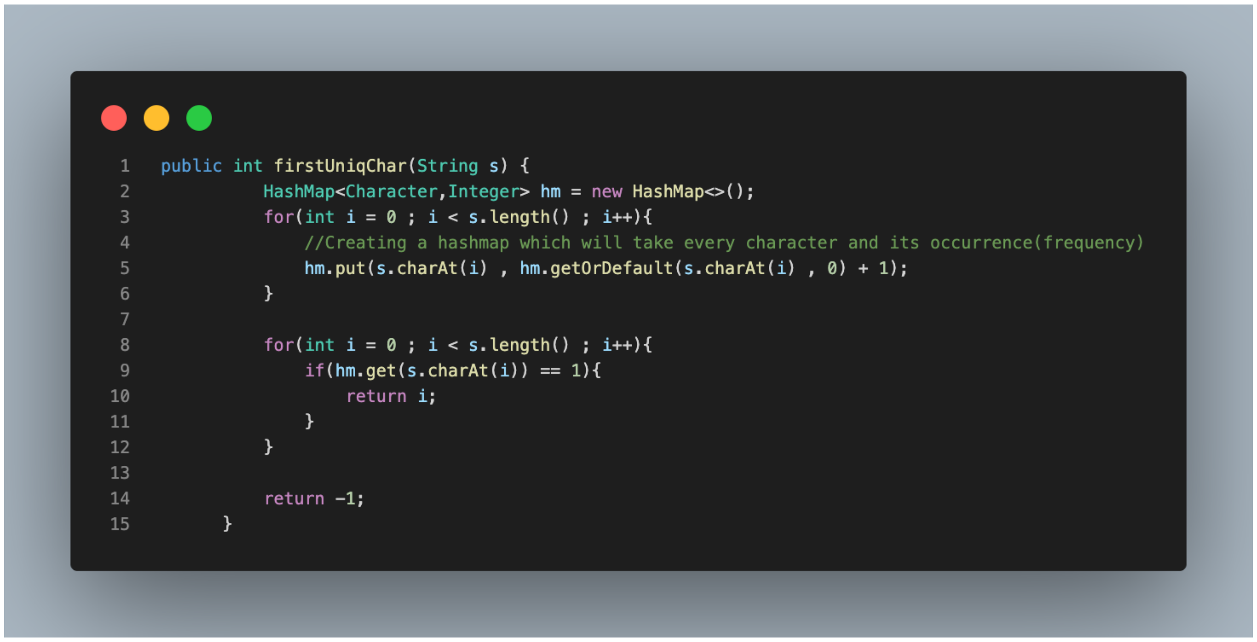
P8. First Unique Character in a String

```
Example 1:
Input: s = "leetcode"
Output: 0

Example 2:
Input: s = "loveleetcode"
Output: 2
```

Approach :

1. Create a HashMap which contains character and it's frequency
2. Then iterate to all the characters and which character's frequency is 1, return index of that character
3. Else return -1

HashMap : Time :O(N)
Space : O(N)



```java
public int firstUniqChar(String s) {
        HashMap<Character,Integer> hm = new HashMap<>();
        for(int i = 0 ; i < s.length() ; i++){
            //Creating a hashmap which will take every character and its occurrence(frequency)
            hm.put(s.charAt(i) , hm.getOrDefault(s.charAt(i) , 0) + 1);
        }

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

        return -1;
    }
```

P9. Find Common Characters (DOUBT)

```
Example 1:
Input: words = ["bella","label","roller"]
Output: ["e","l","l"]

Example 2:
Input: words = ["cool","lock","cook"]
```

```
Output: ["c","o"]
```

Approach :

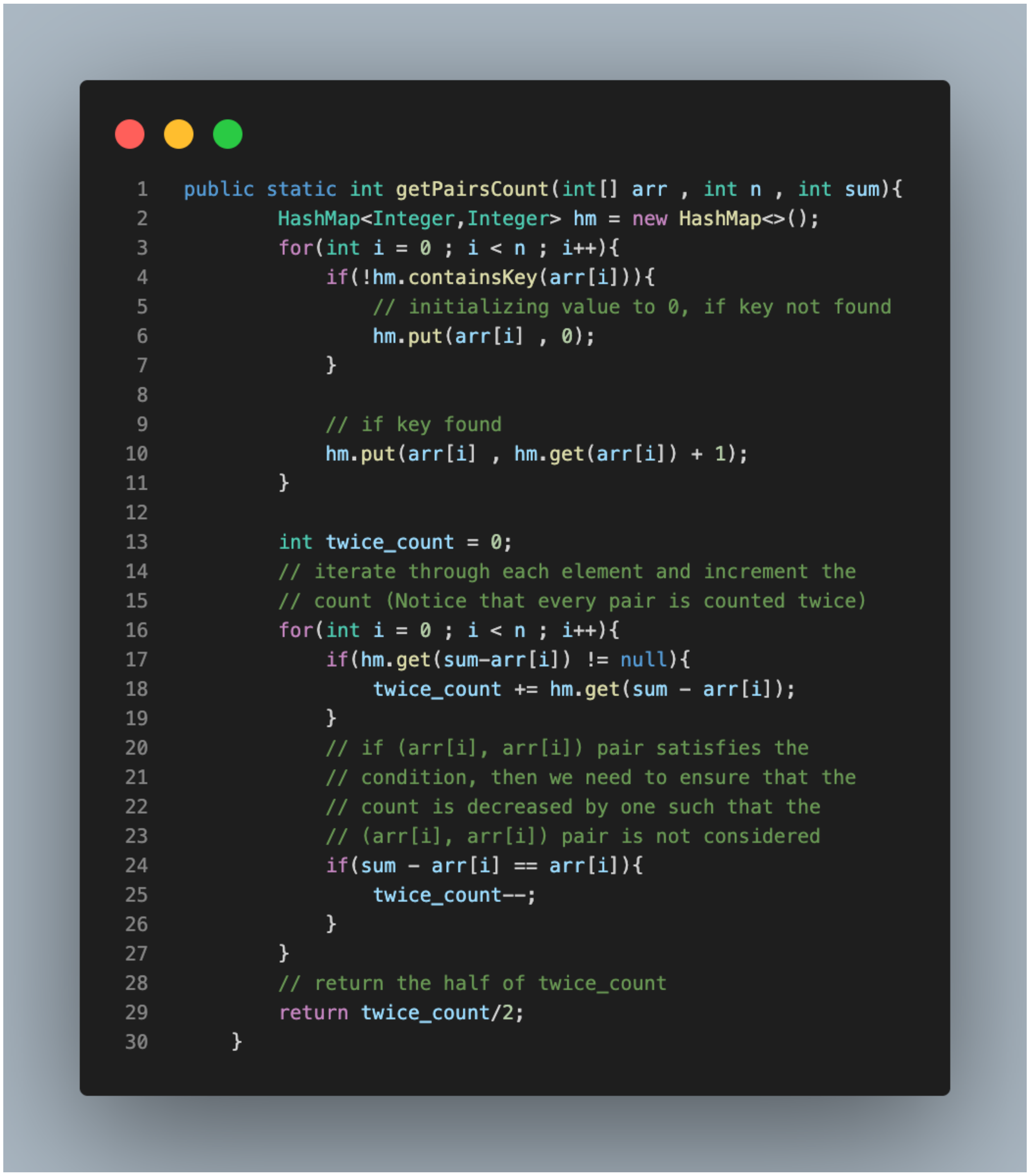
P10. Count pairs with given sum

```
Input:  arr[] = {1, 5, 7, -1}, sum = 6
Output:  2
Explanation: Pairs with sum 6 are (1, 5) and (7, -1).

Input:  arr[] = {10, 12, 10, 15, -1, 7, 6, 5, 4, 2, 1, 1, 1}, sum = 11
Output:  9
Explanation: Pairs with sum 11 are (10, 1), (10, 1), (10, 1), (12, -1), (10, 1), (10, 1), (10, 1), (7,
4), (6, 5).
```

Approach 1:

```
Given arr[] = {1, 5, 7, -1}, sum = 6

1. Store the frequency of every element:
     freq[arr[i]] = freq[arr[i]] + 1
     freq[1] : 1
     freq[5] : 1
     freq[7] : 1
     freq[-1] : 1
2. Initialise a variable count with 0 to find the required count of pairs
3. At index = 0: freq[sum - arr[0]] = freq[6 - 1] = freq[5] = 1
count = 1
4. At index = 1: freq[sum - arr[1]] = freq[6 - 5] = freq[1] = 1
count = 2
5. At index = 2: freq[sum - arr[2]] = freq[6 - 7] = freq[-1] = 1
count = 3
6. At index = 3: freq[sum - arr[3]] = freq[6 - (-1)] = freq[7] = 1
count = 4
7. The above also contains repeated pairs from front and last, i.e. pair (a, b) and (b, a)
are considered as different pairs till now.
Therefore, we will reduce the count by half to determine the count of unique pairs.
count = count / 2 = 2
8. Therefore, required Number of pairs with given sum = 2
```

HashMap : Time : O(N)
Space : O(N)



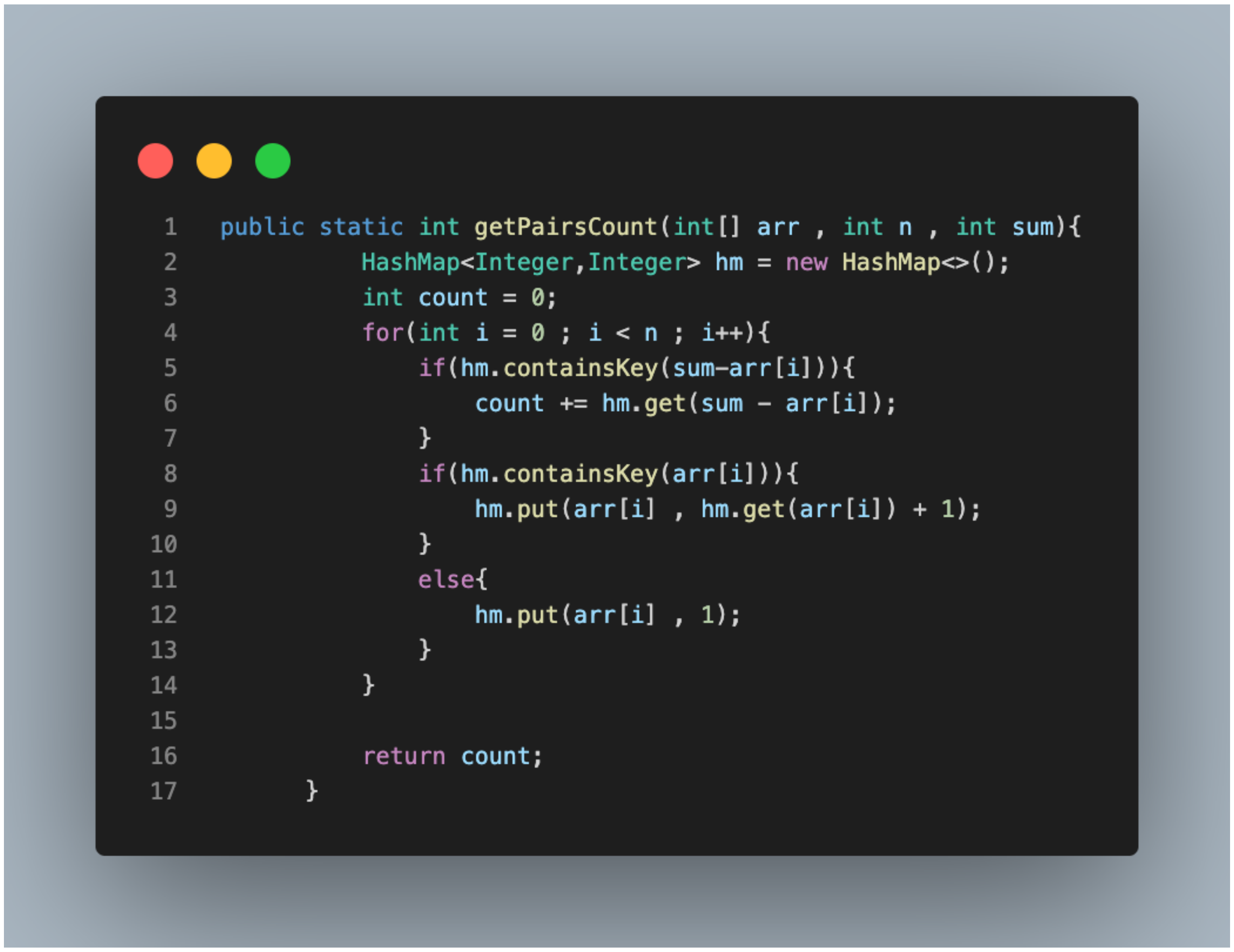
```java
public static int getPairsCount(int[] arr , int n , int sum){
        HashMap<Integer,Integer> hm = new HashMap<>();
        for(int i = 0 ; i < n ; i++){
            if(!hm.containsKey(arr[i])){
                // initializing value to 0, if key not found
                hm.put(arr[i] , 0);
            }

            // if key found
            hm.put(arr[i] , hm.get(arr[i]) + 1);
        }

        int twice_count = 0;
        // iterate through each element and increment the
        // count (Notice that every pair is counted twice)
        for(int i = 0 ; i < n ; i++){
            if(hm.get(sum-arr[i]) != null){
                twice_count += hm.get(sum - arr[i]);
            }
            // if (arr[i], arr[i]) pair satisfies the
            // condition, then we need to ensure that the
            // count is decreased by one such that the
            // (arr[i], arr[i]) pair is not considered
            if(sum - arr[i] == arr[i]){
                twice_count--;
            }
        }
        // return the half of twice_count
        return twice_count/2;
    }
```

Approach 2 : Hashing in Single loop (Better Approach)

```
Given arr[] = {1, 5, 7, -1}, sum = 6


count = 0
```

```
At index = 0: freq[sum - arr[0]] = freq[6 - 1] = freq[5] = 0
count = 0
freq[arr[0]] = freq[1] = 1

At index = 1: freq[sum - arr[1]] = freq[6 - 5] = freq[1] = 1
count = 1
freq[arr[1]] = freq[5] = 1

At index = 2: freq[sum - arr[2]] = freq[6 - 7] = freq[-1] = 0
count = 1
freq[arr[2]] = freq[7] = 1

At index = 3: freq[sum - arr[3]] = freq[6 - (-1)] = freq[7] = 1
count = 2
freq[arr[3]] = freq[-1] = 1

count = 2

Number of pairs  = 2

Follow the steps below to solve the given problem:

Create a map to store the frequency of each number in the array.
Check if (sum - arr[i]) is present in the map, if present then increment the count variable by its
frequency.
After traversal is over, return the count.
```

HashMap : Time : O(N)
Space : O(N)



```
public static int getPairsCount(int[] arr , int n , int sum){
            HashMap<Integer,Integer> hm = new HashMap<>();
            int count = 0;
            for(int i = 0 ; i < n ; i++){
                if(hm.containsKey(sum-arr[i])){
                    count += hm.get(sum - arr[i]);
                }
                if(hm.containsKey(arr[i])){
                    hm.put(arr[i] , hm.get(arr[i]) + 1);
                }
                else{
                    hm.put(arr[i] , 1);
                }
            }

            return count;
        }
```