Problem statement:

> Given an array containing **N** integers and a positive integer **K**, find the length of the longest sub array with sum of the elements divisible by the given value **K**.

Example:

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.
```

Approach:

⟶ Here the question looks like we have to use sliding window method But here we need maximum length of subarray so this method will take `O(n^2)` which is not good.

So after that i calculated prefix sum for all the elements and it looks like this:

```
Arr[] = [2,7,6,1,4,5]
Prefix[] = [2,9,15,16,20,25]
```

we can't also do sliding window in `prefix[]` because ig it will not help.

So then i calculated remainders for all prefix sum and it looks like this:

```
Prefix[] = [2,9,15,16,20,25]
Remainder = [2,0,0,1,2,1]
```

⟶ Here we can see if we take the length between 2 same remainders then it will be the subarray with sum divisible by K. so now we just have to count longest length.

⟶ Now one more problem is that if we have negative numbers then remainder will be negative too. So to handle that we will add k into that to make it positive

We will use `unordered_map` to store the remainders.

At last we will return the longest subarray length.

Code

```
int longSubarrWthSumDivByK(int arr[], int n, int k)
        {
            int sum = 0;
            int ans = 0;
            unordered_map<int,int>mp;
            for(int i=0;i<n;i++){
                sum += arr[i]; // Finding prefix sum
                int r = sum % k; // Finding remainder
                if(r < 0){ // If remainder is negative then add k
into it
                    r += k;
                }
                if(r == 0){ // If remainder is 0 means the current
sum is divisible by k so take the length which we can get with i+1
and update with ans.
                    ans = max(ans,i+1);
                }
                    // Find the remainder in map and if it's
```

```
already present then take the distance between the previous
remainder's index and current index to find length.
                else if(mp.find(r) != mp.end()){
                    ans = max(ans,i-mp[r]);
                        // else add the remainder in map with it's
index.
                }else{
                    mp[r] = i;
                }
            }

            return ans;
        }
```