ARRAY

12 December 2021 19:05

Find Unique

You have been given an integer array/list(ARR) of size N. Where N is equal to [2M + 1].
Now, in the given array/list, 'M' numbers are present twice and one number is present only once.
You need to find and return that number which is unique in the array/list.
Note:
Unique element is always present in the array/list according to the given condition.
Input format :
The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

First line of each test case or query contains an integer 'N' representing the size of the array/list.

Second line contains 'N' single space separated integers representing the elements in the array/list.
Output Format :
For each test case, print the unique element present in the array.

Output for every test case will be printed in a separate line.
Constraints :
1 <= t <= 10^2
0 <= N <= 10^3

Time Limit: 1 sec

Sample Input 1:

```
1
7
2 3 1 6 3 6 2
```

Sample Output 1:

```
1
```

Sample Input 2:

```
2
5
2 4 7 2 7
9
1 3 1 3 6 6 7 10 7
```

Sample Output 2:

```
4
10
```

Find Unique

Problem Description: Given an integer array of size 2N + 1. In this given array, N numbers are present twice and one number is present only once in the array. You need to find and return that number which is unique in the array.
For example: size= 7
arr[] = 2 3 1 6 3 6 2
Output: 1

How to approach?
Approach 1: For each element of the input array, you need to run a loop on the whole array, to check whether the same element exists or not, if it exists then we check for the next element, else we will return that element, as we found the unique element.

Pseudo Code for this approach:

```
Function FindUnique:
    For i = 0 to i less than size:
        For j = 0 to j less than size:
            If i not equal to j:
                If arr[i] is equal to arr[j]:
                    break
        If j is equal to size:
            return arr[i]
    return minus infinity
```

Time Complexity for this approach: Time complexity for this approach is O(n^2), which is not good, hence we move to the next approach.

Approach 2: We can Use Sorting to solve the problem in O(nLogn) time. The idea is simple, first sort the array, so that all occurrences of every element become consecutive. Once the occurrences become consecutive, we can traverse the sorted array and print the unique element in O(n) time.

Approach 3: The best solution is to use XOR. Taking XOR of all the elements present in the array, gives us the unique element present in the array.

This approach uses 2 properties of XOR:

1. XOR of a number with itself is 0.
2. XOR of a number with 0 is number itself.

Let us understand this approach with the help of an example:
arr[]= 2 3 1 6 3 6 2
Taking their xor:
Answer = 2 ^ 3 ^ 1 ^ 6 ^ 3 ^ 6 ^ 2
Now XOR is associative and commutative, so I can write it as:
Answer = (2^2) ^ (3^3) ^ 1 ^ (6 ^ 6)
= 0 ^ 0 ^ 1 ^ 0
= 1
Time complexity of this solution is O(n).

Pseudo Code for this approach:

```
Function findunique:
    Answer=0
    For i=0 to i less than size:
        answer=answer xor input[i]
   Return answer
```

Code:

```cpp
int findUnique(int a[], int n, int k) {
        unordered_map<int,int>um;
        for(int i=0;i<n;i++){
            um[a[i]]++;
        }
        for(int i=0;i<n;i++){
            if(um[a[i]]%k!=0){
                return a[i];
            }
        }
        return -1;
    }
```