GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

Login

| Home | Algorithms | DS | GATE | Interview Corner | Q&A | C | C++ | Java | Books | Contribute | Ask a Q | About |

| Array | Bit Magic | C/C++ | Articles | GFacts | Linked List | MCQ | Misc | Output | String | Tree | Graph |

Find the maximum repeating number in O(n) time and O(1) extra space

Given an array of size *n*, the array contains numbers in range from 0 to *k-1* where *k* is a positive integer and *k <= n.* Find the maximum repeating number in this array. For example, let *k* be 10 the given array be *arr[]* = {1, 2, 2, 2, 0, 2, 0, 2, 3, 8, 0, 9, 2, 3}, the maximum repeating number would be 2. Expected time complexity is *O(n)* and extra space allowed is *O(1)*. Modifications to array are allowed.

The **naive approach** is to run two loops, the outer loop picks an element one by one, the inner loop counts number of occurrences of the picked element. Finally return the element with maximum count. Time complexity of this approach is *O(n^2)*.

A **better approach** is to create a count array of size k and initialize all elements of *count[]* as 0. Iterate through all elements of input array, and for every element *arr[i]*, increment *count[arr[i]]*. Finally, iterate through *count[]* and return the index with maximum value. This approach takes O(n) time, but requires O(k) space.

Following is the *O(n) time and O(1) extra space* approach. Let us understand the approach with a simple example where *arr[]* = {2, 3, 3, 5, 3, 4, 1, 7}, *k* = 8, *n* = 8 (number of elements in arr[]).

1) Iterate though input array *arr[]*, for every element *arr[i]*, increment *arr[arr[i]%k]* by *k* (*arr[]* becomes {2, 11, 11, 29, 11, 12, 1, 15 })

2) Find the maximum value in the modified array (maximum value is 29). Index of the maximum value is the maximum repeating element (index of 29 is 3).

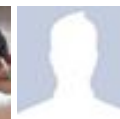**3)** If we want to get the original array back, we can iterate through the array one more time and

do *arr[i] = arr[i] % k* where *i* varies from 0 to *n-1*.

How does the above algorithm work? Since we use *arr[i]%k* as index and add value *k* at the index *arr[i]%k*, the index which is equal to maximum repeating element will have the maximum value in the end. Note that *k* is added maximum number of times at the index equal to maximum repeating element and all array elements are smaller than *k*.

Following is C++ implementation of the above algorithm.

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

```cpp
// Returns maximum repeating element in arr[0..n-1].
// The array elements are in range from 0 to k-1
int maxRepeating(int* arr, int n, int k)
{
    // Iterate though input array, for every element
    // arr[i], increment arr[arr[i]%k] by k
    for (int i = 0; i< n; i++)
        arr[arr[i]%k] += k;

    // Find index of the maximum repeating element
    int max = arr[0], result = 0;
    for (int i = 1; i < n; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
            result = i;
        }
    }

    /* Uncomment this code to get the original array back
       for (int i = 0; i< n; i++)
          arr[i] = arr[i]%k; */

    // Return index of the maximum element
    return result;
}
```

```cpp
// Driver program to test above function
int main()
{
    int arr[] = {2, 3, 3, 5, 3, 4, 1, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
```

Popular Posts

All permutations of a given string

Memory Layout of C Programs

Understanding "extern" keyword in C

Median of two sorted arrays

Tree traversal without recursion and without stack!

Structure Member Alignment, Padding and Data Packing

Intersection point of two Linked Lists

Lowest Common Ancestor in a BST.

Check if a binary tree is BST or not

Sorted Linked List to Balanced BST

```cpp
    int k = 8;

    cout << "The maximum repeating number is " <<
        maxRepeating(arr, n, k) << endl;

    return 0;
}
```
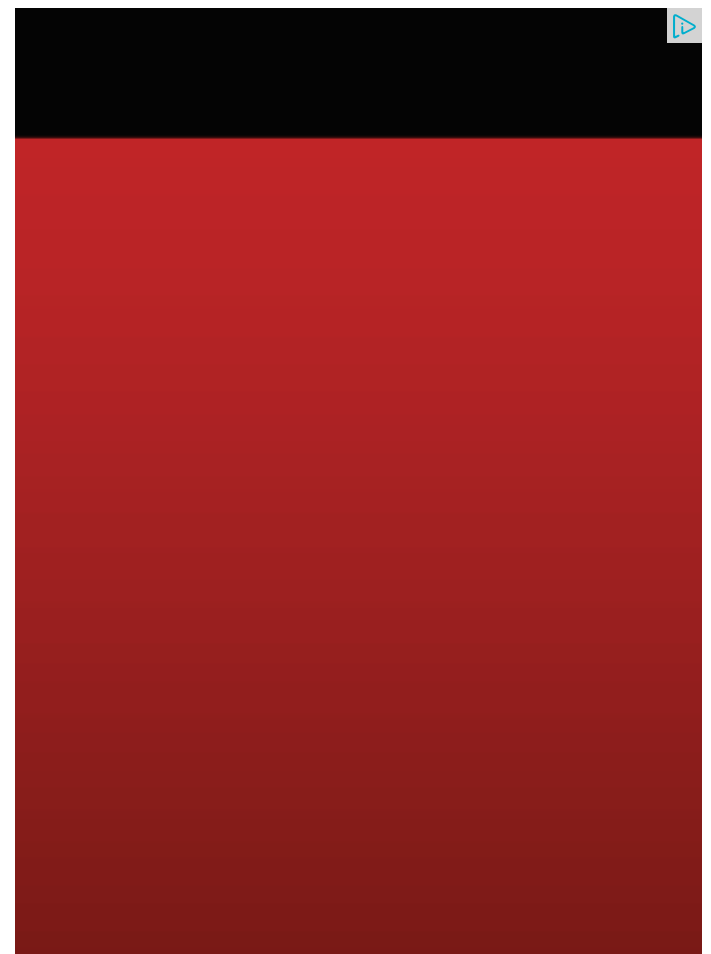
Output:
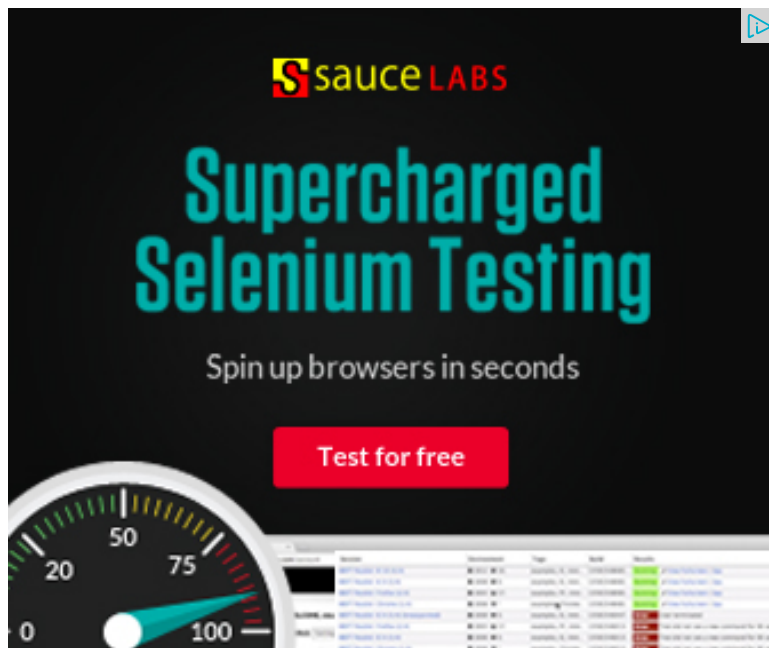
```
The maximum repeating number is 3
```

Exercise:
The above solution prints only one repeating element and doesn't work if we want to print all maximum repeating elements. For example, if the input array is {2, 3, 2, 3}, the above solution will print only 3. What if we need to print both of 2 and 3 as both of them occur maximum number of times. Write a O(n) time and O(1) extra space function that prints all maximum repeating elements. (Hint: We can use maximum quotient arr[i]/n instead of maximum value in step 2).

Note that the above solutions may cause overflow if adding k repeatedly makes the value more than INT_MAX.

This article is compiled by Ashish Anand and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

705 Subscribe

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

20   **Tweet** 5   2

Writing code in comment? Please use **ideone.com** and share the link here.

Recent Comments

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 2 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 5 minutes ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 30 minutes ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 32 minutes ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 57 minutes ago

newCoder3006 Code without using while loop. We can do it...

► JavaScript Array

► Java Array

► C++ Array

► Array Max

► Memory Array

► An Array

► Python Array

► Code Number

► Print a Number

@geeksforgeeks, **Some rights reserved**   **Contact Us!**   Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team