Count distinct elements in every window of size k

Given an array of size n and an integer k, return the of count of distinct numbers in all windows of size k.

Example:

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

Explanation:
First window is {1, 2, 1, 3}, count of distinct numbers is 3
Second window is {2, 1, 3, 4} count of distinct numbers is 4
Third window is {1, 3, 4, 2} count of distinct numbers is 4
Fourth window is {3, 4, 2, 3} count of distinct numbers is 3
```

We strongly recommend you to minimize your browser and try this yourself first.

Source: Microsoft Interview Question

A **Simple Solution** is to traverse the given array, consider every window in it and count distinct elements in the window. Below is C++ implementation of simple solution.

```
// Simple C++ program to count distinct elements in every
// window of size k
#include <iostream>
using namespace std;

// Counts distinct elements in window of size k
int countWindowDistinct(int win[], int k)
{
    int dist_count = 0;

    // Traverse the
    for (int i=0; i<k; i++)
    {
        // Check if element arr[i] exists in arr[0..i-1]
        int j;
        for (j=0; j<i; j++)
           if (win[i] == win[j])
              break;
        if (j==i)
            dist_count++;
    }
    return dist_count;
}

// Counts distinct elements in all windows of size k
void countDistinct(int arr[], int n, int k)
{
    // Traverse through every window
    for (int i=0; i<=n-k; i++)
       cout << countWindowDistinct(arr+i, k) << endl;
}

// Driver program
int main()
{
    int arr[] = {1, 2, 1, 3, 4, 2, 3},  k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    countDistinct(arr, n, k);
    return 0;
}
```

Output:

```
3
4
4
3
```

Time complexity of the above solution is $O(nk^2)$. We can improve time complexity to O(nkLok) by modifying countWindowDistinct() to use sorting. The function can further be optimized to use hashing to find distinct elements in a window. With hashing the time complexity becomes O(nk). Below is a different approach that works in O(n) time.

An **Efficient Solution** is to use the count of previous window, while sliding the window. The idea is to create a hash map that stores elements of current widow. When we slide the window, we remove an element from hash and add an element. We also keep track of distinct elements. Below is algorithm.

1) Create an empty hash map. Let hash map be hM

2) Initialize distinct element count 'dist_count' as 0.

3) Traverse through first window and insert elements of first window to hM. The elements are used as key and their counts as value in hM. Also keep updating 'dist_count'

4) Print 'dist_count' for first window.

3) Traverse through remaining array (or other windows).
....a) Remove the first element of previous window.
.....If the removed element appeared only once
.....remove it from hM and do "dist_count–"
.....Else (appeared multiple times in hM)

……..decrement its count in hM

….a) Add the current element (last element of new window)
…….If the added element is not present in hM
……..add it to hM and do “dist_count++”
…….Else (the added element appeared multiple times)
……..increment its count in hM

Below is implementation of above approach.

Java

```
// An efficient Java program to count distinct elements in
// every window of size k
import java.util.HashMap;

class CountDistinctWindow
{
    static void countDistinct(int arr[], int k)
    {
        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM =
                       new HashMap<Integer, Integer>();

        // initialize distinct element  count for
        // current window
        int dist_count = 0;

        // Traverse the first window and store count
        // of every element in hash map
        for (int i = 0; i < k; i++)
        {
            if (hM.get(arr[i]) == null)
            {
                hM.put(arr[i], 1);
                dist_count++;
            }
            else
            {
               int count = hM.get(arr[i]);
               hM.put(arr[i], count+1);
            }
        }

        // Print count of first window
        System.out.println(dist_count);

        // Traverse through the remaining array
        for (int i = k; i < arr.length; i++)
        {

            // Remove first element of previous window
            // If there was only one occurrence, then
            // reduce distinct count.
            if (hM.get(arr[i-k]) == 1)
            {
                hM.remove(arr[i-k]);
                dist_count--;
            }
            else // reduce count of the removed element
            {
               int count = hM.get(arr[i-k]);
               hM.put(arr[i-k], count-1);
            }

            // Add new element of current window
            // If this element appears first time,
            // increment distinct element count
            if (hM.get(arr[i]) == null)
            {
```

```
                hM.put(arr[i], 1);
                dist_count++;
            }
            else // Increment distinct element count
            {
               int count = hM.get(arr[i]);
               hM.put(arr[i], count+1);
            }

           // Print count of current window
            System.out.println(dist_count);
        }
    }

    // Driver method
    public static void main(String arg[])
    {
        int arr[] =  {1, 2, 1, 3, 4, 2, 3};
        int k = 4;
        countDistinct(arr, k);
    }
}
```

C++

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

void countDistinct(int arr[], int k, int n)
{
    // Creates an empty hashmap hm
    map<int, int> hm;

    // initialize distinct element count for current window
    int dist_count = 0;

    // Traverse the first window and store count
    // of every element in hash map
    for (int i = 0; i < k; i++)
    {
       if (hm[arr[i]] == 0)
       {
           dist_count++;
       }
    hm[arr[i]] += 1;
    }

   // Print count of first window
  cout << dist_count << endl;

   // Traverse through the remaining array
   for (int i = k; i < n; i++)
   {
     // Remove first element of previous window
     // If there was only one occurrence, then reduce distinct count.
     if (hm[arr[i-k]] == 1)
    {
        dist_count--;
    }
   // reduce count of the removed element
   hm[arr[i-k]] -= 1;

   // Add new element of current window
   // If this element appears first time,
   // increment distinct element count

  if (hm[arr[i]] == 0)
  {
     dist_count++;
  }
  hm[arr[i]] += 1;

  // Print count of current window
  cout << dist_count << endl;
  }
}

int main()
{
   int arr[] = {1, 2, 1, 3, 4, 2, 3};
   int size = sizeof(arr)/sizeof(arr[0]);
   int k = 4;
   countDistinct(arr, k, size);

   return 0;
}
//This solution is contributed by Aditya Goel
```

Output:

```
3
4
4
3
```

Time complexity of the above solution is O(n).