Count of subarrays having exactly K distinct elements

Difficulty Level : Hard • Last Updated : 20 Dec, 2021

Given an array **arr[]** of size **N** and an integer **K**. The task is to find the count of subarrays such that each subarray has exactly **K** distinct elements.

Examples:

Input: *arr[] = {2, 1, 2, 1, 6}, K = 2*
Output: *7*
{2, 1}, {1, 2}, {2, 1}, {1, 6}, {2, 1, 2},
{1, 2, 1} and {2, 1, 2, 1} are the only valid subarrays.

Input: *arr[] = {1, 2, 3, 4, 5}, K = 1*
Output: *5*

Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

different integers is easy. So the idea is to find the count of subarrays with **at most K** different integers, let it be **C(K)**, and the count of subarrays with **at most (K – 1)** different integers, let it be **C(K – 1)** and finally take their difference, **C(K) – C(K – 1)** which is the required answer.

Count of subarrays with at most **K** different elements can be easily calculated through the sliding window technique. The idea is to keep expanding the right boundary of the window till the count of distinct elements in the window is less than or equal to **K** and when the count of distinct elements inside the window becomes more than **K**, start shrinking the window from the left till the count becomes less than or equal to **K**. Also for every expansion, keep counting the subarrays as **right – left + 1** where **right** and **left** are the boundaries of the current window.

Below is the implementation of the above approach:

C++

```
// C++ implementation of the approach
#include<bits/stdc++.h>
#include<map>
using namespace std;


// Function to return the count of subarrays
// with at most K distinct elements using
// the sliding window technique
int atMostK(int arr[], int n, int k)
{

    // To store the result
    int count = 0;
```

```
    int right = 0;

    // Map to keep track of number of distinct
    // elements in the current window
    unordered_map<int,int> map;
    // Loop to calculate the count
    while (right < n) {

        // Calculating the frequency of each
        // element in the current window
        if (map.find(arr[right])==map.end())
            map[arr[right]]=0;
        map[arr[right]]++;

        // Shrinking the window from left if the
        // count of distinct elements exceeds K
        while (map.size() > k) {
            map[arr[left]]= map[arr[left]] - 1;
            if (map[arr[left]] == 0)
                map.erase(arr[left]);
            left++;
        }

        // Adding the count of subarrays with at most
        // K distinct elements in the current window
        count += right - left + 1;
        right++;
    }
    return count;
}

// Function to return the count of subarrays
// with exactly K distinct elements
int exactlyK(int arr[], int n, int k)
{

    // Count of subarrays with exactly k distinct
    // elements is equal to the difference of the
    // count of subarrays with at most K distinct
    // elements and the count of subararys with
    // at most (K - 1) distinct elements
```

```
int main()
{
    int arr[] = { 2, 1, 2, 1, 6 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 2;

    cout<<(exactlyK(arr, n, k));
}
```

Java

```
// Java implementation of the approach
import java.util.*;

public class GfG {

    // Function to return the count of subarrays
    // with at most K distinct elements using
    // the sliding window technique
    private static int atMostK(int arr[], int n, int k)
    {

        // To store the result
        int count = 0;

        // Left boundary of window
        int left = 0;

        // Right boundary of window
        int right = 0;

        // Map to keep track of number of distinct
        // elements in the current window
        HashMap<Integer, Integer> map = new HashMap<>();

        // Loop to calculate the count
        while (right < n) {

            // Calculating the frequency of each
```

```
            // count of distinct elements exceeds K
            while (map.size() > k) {
                map.put(arr[left], map.get(arr[left]) - 1);
                if (map.get(arr[left]) == 0)
                    map.remove(arr[left]);
                left++;
            }

            // Adding the count of subarrays with at most
            // K distinct elements in the current window
            count += right - left + 1;
            right++;
        }
        return count;
    }

    // Function to return the count of subarrays
    // with exactly K distinct elements
    private static int exactlyK(int arr[], int n, int k)
    {

        // Count of subarrays with exactly k distinct
        // elements is equal to the difference of the
        // count of subarrays with at most K distinct
        // elements and the count of subararys with
        // at most (K - 1) distinct elements
        return (atMostK(arr, n, k)
                - atMostK(arr, n, k - 1));
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 2, 1, 2, 1, 6 };
        int n = arr.length;
        int k = 2;

        System.out.print(exactlyK(arr, n, k));
    }
}
```

```python
# Python3 implementation of the above approach

# Function to return the count of subarrays
# with at most K distinct elements using
# the sliding window technique


def atMostK(arr, n, k):

    # To store the result
    count = 0

    # Left boundary of window
    left = 0

    # Right boundary of window
    right = 0

    # Map to keep track of number of distinct
    # elements in the current window
    map = {}

    # Loop to calculate the count
    while(right < n):

        if arr[right] not in map:
            map[arr[right]] = 0

        # Calculating the frequency of each
        # element in the current window
        map[arr[right]] += 1

        # Shrinking the window from left if the
        # count of distinct elements exceeds K
        while(len(map) > k):

            if arr[left] not in map:
                map[arr[left]] = 0

            map[arr[left]] -= 1
```

```
        # Adding the count of subarrays with at most
        # K distinct elements in the current window
        count += right - left + 1
        right += 1

    return count

# Function to return the count of subarrays
# with exactly K distinct elements


def exactlyK(arr, n, k):

    # Count of subarrays with exactly k distinct
    # elements is equal to the difference of the
    # count of subarrays with at most K distinct
    # elements and the count of subararys with
    # at most (K - 1) distinct elements
    return (atMostK(arr, n, k) -
            atMostK(arr, n, k - 1))


# Driver code
if __name__ == "__main__":
    arr = [2, 1, 2, 1, 6]
    n = len(arr)
    k = 2

    print(exactlyK(arr, n, k))

# This code is contributed by AnkitRai01
```

C#

```
// C# implementation of the approach
using System;
using System.Collections.Generic;

class GfG {
```

```
private static int atMostK(int[] arr, int n, int k)
{

    // To store the result
    int count = 0;

    // Left boundary of window
    int left = 0;

    // Right boundary of window
    int right = 0;

    // Map to keep track of number of distinct
    // elements in the current window
    Dictionary<int, int> map
        = new Dictionary<int, int>();

    // Loop to calculate the count
    while (right < n) {

        // Calculating the frequency of each
        // element in the current window
        if (map.ContainsKey(arr[right]))
            map[arr[right]] = map[arr[right]] + 1;
        else
            map.Add(arr[right], 1);

        // Shrinking the window from left if the
        // count of distinct elements exceeds K
        while (map.Count > k) {
            if (map.ContainsKey(arr[left])) {
                map[arr[left]] = map[arr[left]] - 1;
                if (map[arr[left]] == 0)
                    map.Remove(arr[left]);
            }
            left++;
        }

        // Adding the count of subarrays with at most
        // K distinct elements in the current window
        count += right - left + 1;
```

```
    // Function to return the count of subarrays
    // with exactly K distinct elements
    private static int exactlyK(int[] arr, int n, int k)
    {

        // Count of subarrays with exactly k distinct
        // elements is equal to the difference of the
        // count of subarrays with at most K distinct
        // elements and the count of subararys with
        // at most (K - 1) distinct elements
        return (atMostK(arr, n, k)
                - atMostK(arr, n, k - 1));
    }

    // Driver code
    public static void Main(String[] args)
    {
        int[] arr = { 2, 1, 2, 1, 6 };
        int n = arr.Length;
        int k = 2;

        Console.Write(exactlyK(arr, n, k));
    }
}

// This code is contributed by 29AjayKumar
```

Javascript

```
<script>

// Javascript implementation of the approach

// Function to return the count of subarrays
// with at most K distinct elements using
// the sliding window technique
function atMostK(arr, n, k)
{
    // To store the result
```

```
    // Right boundary of window
    let right = 0;

    // Map to keep track of number of distinct
    // elements in the current window
    let map = new Map();

    // Loop to calculate the count
    while (right < n)
    {

        // Calculating the frequency of each
        // element in the current window
        if (map.has(arr[right]))
            map.set(arr[right],
            map.get(arr[right]) + 1);
        else
            map.set(arr[right], 1);

        // Shrinking the window from left if the
        // count of distinct elements exceeds K
        while (map.size > k)
        {
            map.set(arr[left], map.get(arr[left]) - 1);
            if (map.get(arr[left]) == 0)
                map.delete(arr[left]);

            left++;
        }

        // Adding the count of subarrays with at most
        // K distinct elements in the current window
        count += right - left + 1;
        right++;
    }
    return count;
}

// Function to return the count of subarrays
// with exactly K distinct elements
function exactlyK(arr, n, k)
```

```
    // count of subarrays with at most K distinct
    // elements and the count of subararys with
    // at most (K - 1) distinct elements
    return (atMostK(arr, n, k) -
            atMostK(arr, n, k - 1));
}

// Driver code
let arr = [ 2, 1, 2, 1, 6 ];
let n = arr.length;
let k = 2;

document.write(exactlyK(arr, n, k));

// This code is contributed by avanitrachhadiya2155

</script>
```

Output

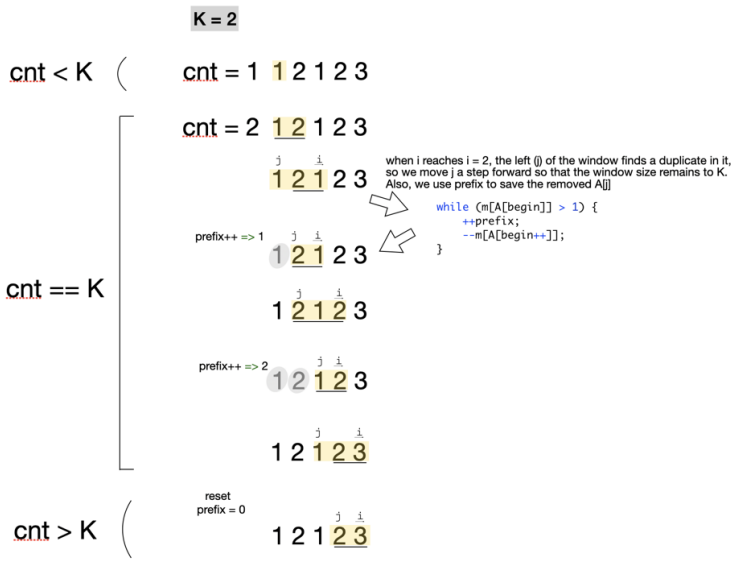
```
 7
```

Time Complexity: O(N)

Space Complexity: O(N)

Another Approach: When you move the right cursor, keep tracking whether we have reach a count of K distinct integers, if yes, we process left cursor, here is how we process left cursor:

- check whether the element pointed by the left cursor is duplicated in the window, if yes, we remove it, and use a variable (e.g. prefix) to record that we have removed an element from the window). keep this process until we reduce the window size from to exactly K. now we can calculate the number of the valid good array as res += prefix;
- after process left cursor and all the stuff, the outer loop will continue and the right cursor will move forward, and then the window size will



Below is the implementation of the above approach:

C++

```
// C++ program to calculate number
// of subarrays with distinct elements of size k
#include <bits/stdc++.h>
#include <map>
#include <vector>
using namespace std;

int subarraysWithKDistinct(vector<int>& A, int K)
{

    // declare a map for the frequency
    unordered_map<int, int> mapp;
    int begin = 0, end = 0, prefix = 0, cnt = 0;
    int res = 0;

    // traverse the array
```

```
            cnt++;
        }
        end++;
        if (cnt > K)
        {
            mapp[A[begin]]--;
            begin++;
            cnt--;
            prefix = 0;
        }

        // loop until mapp[A[begin]] > 1
        while (mapp[A[begin]] > 1)
        {
            mapp[A[begin]]--;
            begin++;
            prefix++;
        }
        if (cnt == K)
        {
            res += prefix + 1;
        }
    }

    // return the final count
    return res;
}
// Driver code
int main()
{
    vector<int> arr{ 2, 1, 2, 1, 6 };
    int k = 2;

     // Function call
    cout << (subarraysWithKDistinct(arr, k));
}
// This code is contributed by Harman Singh
```

Java

```
class GFG
{
  static int subarraysWithKDistinct(int A[], int K)
  {

    // declare a map for the frequency
    HashMap<Integer, Integer> mapp = new HashMap<>();
    int begin = 0, end = 0, prefix = 0, cnt = 0;
    int res = 0;

    // traverse the array
    while (end < A.length)
    {

      // increase the frequency
      if(mapp.containsKey(A[end]))
      {
        mapp.put(A[end], mapp.get(A[end]) + 1);
      }
      else
      {
        mapp.put(A[end], 1);
      }
      if (mapp.get(A[end]) == 1)
      {
        cnt++;
      }
      end++;
      if (cnt > K)
      {
        if(mapp.containsKey(A[begin]))
        {
          mapp.put(A[begin], mapp.get(A[begin]) - 1);
        }
        else
        {
          mapp.put(A[begin], -1);
        }
        begin++;
        cnt--;
        prefix = 0;
```

```
        {
          if(mapp.containsKey(A[begin]))
          {
            mapp.put(A[begin], mapp.get(A[begin]) - 1);
          }
          else
          {
            mapp.put(A[begin], -1);
          }
          begin++;
          prefix++;
        }
        if (cnt == K)
        {
          res += prefix + 1;
        }
      }

      // return the final count
      return res;
    }

    // Driver code
    public static void main(String[] args)
    {
      int arr[] = { 2, 1, 2, 1, 6 };
      int k = 2;

      // Function call
      System.out.println(subarraysWithKDistinct(arr, k));
    }
}

// This code is contributed by divyeshrabadiya07
```

Python3

```
# Python3 program to calculate number of
# subarrays with distinct elements of size k
def subarraysWithKDistinct(A, K):
```

```
    res = 0

    # Traverse the array
    while (end < len(A)):

        # Increase the frequency
        mapp[A[end]] = mapp.get(A[end], 0) + 1

        if (mapp[A[end]] == 1):
            cnt += 1

        end += 1

        if (cnt > K):
            mapp[A[begin]] -= 1
            begin += 1
            cnt -= 1
            prefix = 0

        # Loop until mapp[A[begin]] > 1
        while (mapp[A[begin]] > 1):
            mapp[A[begin]] -= 1
            begin += 1
            prefix += 1

        if (cnt == K):
            res += prefix + 1

    # Return the final count
    return res

# Driver code
if __name__ == '__main__':

    arr = [ 2, 1, 2, 1, 6 ]
    k = 2

    # Function call
    print (subarraysWithKDistinct(arr, k))

# This code is contributed by Mohit kumar
```

```
// C# program to calculate number
// of subarrays with distinct elements of size k
using System;
using System.Collections.Generic;
class GFG {

    static int subarraysWithKDistinct(List<int> A, int K)
    {

        // declare a map for the frequency
        Dictionary<int, int> mapp = new Dictionary<int, int>();
        int begin = 0, end = 0, prefix = 0, cnt = 0;
        int res = 0;

        // traverse the array
        while (end < A.Count)
        {

            // increase the frequency
            if(mapp.ContainsKey(A[end]))
            {
                mapp[A[end]]++;
            }
            else{
                mapp[A[end]] = 1;
            }
            if (mapp[A[end]] == 1) {
                cnt++;
            }
            end++;
            if (cnt > K)
            {
                if(mapp.ContainsKey(A[begin]))
                {
                    mapp[A[begin]]--;
                }
                else{
                    mapp[A[begin]] = -1;
                }
                begin++;
                cnt  .
```

```
            while (mapp[A[begin]] > 1)
            {
                mapp[A[begin]]--;
                begin++;
                prefix++;
            }
            if (cnt == K)
            {
                res += prefix + 1;
            }
        }

        // return the final count
        return res;
    }

  // Driver code
  static void Main()
  {
    List<int> arr = new List<int>(new int[] { 2, 1, 2, 1, 6 });
    int k = 2;

     // Function call
    Console.Write(subarraysWithKDistinct(arr, k));
  }
}

// This code is contributed by divyesh072019
```

Javascript

```
<script>

// Javascript program to calculate number
// of subarrays with distinct elements of size k
function subarraysWithKDistinct(A, K)
{

    // Declare a map for the frequency
    let mapp = new Map();
```

```
while (end < A.length)
{

    // increase the frequency
    if (mapp.has(A[end]))
    {
        mapp.set(A[end],
        mapp.get(A[end]) + 1);
    }
    else
    {
        mapp.set(A[end], 1);
    }
    if (mapp.get(A[end]) == 1)
    {
        cnt++;
    }
    end++;
    if (cnt > K)
    {
        if (mapp.has(A[begin]))
        {
            mapp.set(A[begin],
            mapp.get(A[begin]) - 1);
        }
        else
        {
            mapp.set(A[begin], -1);
        }
        begin++;
        cnt--;
        prefix = 0;
    }

    // loop until mapp[A[begin]] > 1
    while (mapp.get(A[begin]) > 1)
    {
        if(mapp.has(A[begin]))
        {
            mapp.set(A[begin],
            mapp.get(A[begin]) - 1);
```

```
            }
            begin++;
            prefix++;
        }
        if (cnt == K)
        {
            res += prefix + 1;
        }
    }

    // Return the final count
    return res;
}

// Driver code
let arr = [ 2, 1, 2, 1, 6 ];
let k = 2;

// Function call
document.write(subarraysWithKDistinct(arr, k));

// This code is contributed by rag2127

</script>
```

Output

```
 7
```

Time Complexity: O(N)

Auxiliary Space: O(N)

Previous

Find the length of the longest subarray with atmost K occurrences of the integer X

Next

Java SAX Library

RECOMMENDED ARTICLES

01 **Count subarrays having at least X distinct elements that occur exactly Y times**
25, May 22

02 **Count distinct sequences obtained by replacing all elements of subarrays having equal first and last elements with the first element any number of times**
29, Jun 21

03 **Count subarrays made up of elements having exactly K set bits**
17, Feb 21

04 **Count subarrays having exactly K elements occurring at least twice**
03, Mar 21

05 **Count subarrays having total distinct elements same as original array**
21, Jun 17

06 **Count of subarrays having exactly K prime numbers**
21, Apr 20

07 **Count of subarrays having exactly K perfect square numbers**
12, May 20

08 **Construct an Array having K Subarrays with all distinct elements**
28, Mar 22

Article Contributed By :

ishan_trivedi
@ishan_trivedi

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

GeeksforGeeks
rporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Improved By : 29AjayKumar, ankthon, ukasp, harmansinghh, mohit kumar 29, divyesh072019, divyeshrabadiya07, avanitrachhadiya2155, rag2127, sooda367, pankajsharmagfg, ashutoshsinghgeeksforgeeks, amartyaghoshgfg

Article Tags : sliding-window, subarray, Algorithms, Arrays

Practice Tags : sliding-window, Arrays, Algorithms

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

NodeJS