Find k numbers with most occurrences in the given array

Difficulty Level : Medium • Last Updated : 13 Jul, 2022

Given an array of **n** numbers and a positive integer **k**. The problem is to find **k** numbers with most occurrences, i.e., the top **k** numbers having the maximum frequency. If two numbers have the same frequency then the larger number should be given preference. The numbers should be displayed in decreasing order of their frequencies. It is assumed that the array consists of **k** numbers with most occurrences.

Examples:

> ***Input:***
> *arr[] = {3, 1, 4, 4, 5, 2, 6, 1},*
> *k = 2*
> ***Output:*** *4 1*
> ***Explanation:***
> *Frequency of **4** = 2*
> *Frequency of **1** = 2*
> *These two have the maximum frequency and*
> ***4** is larger than **1***

k = 4

Output: *5 11 7 10*

Explanation:

Frequency of ***5*** *= 3*

Frequency of ***11*** *= 2*

Frequency of ***7*** *= 2*

Frequency of ***10*** *= 1*

These four have the maximum frequency and

5 *is largest among rest.*

Asked in Amazon Interview

Recommended PracticeTop K Frequent Elements in Array - | Try It!

Method 1:

- **Approach:** The thought process should begin from creating a HashMap to store element-frequency pair in the HashMap. HashMap is used to perform insertion and updation in constant time. Then sort the element-frequency pair in decreasing order of frequency. This gives the information about each element and the number of times they are present in the array. To get k elements of the array, print the first k elements of the sorted array.
- **Hashmap:** HashMap is a part of Java's collection since Java 1.2. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs. To access a value one must know its key. HashMap is known as HashMap because it uses a technique called Hashing. Hashing is a technique of converting a

value pairs already explained in HashSet in detail and further articles.

More on HashMap >>

- **Algorithm:**
  1. Create a Hashmap *hm*, to store key-value pair, i.e. element-frequency pair.
  2. Traverse the array from start to end.
  3. For every element in the array update *hm[array[i]]++*
  4. Store the element-frequency pair in a vector and sort the vector in decreasing order of frequency.
  5. Print the first k elements of sorted array.

Below is the Implementation of the above algorithm:

C++

```
// C++ implementation to find k numbers with most
// occurrences in the given array
#include <bits/stdc++.h>

using namespace std;

// comparison function to sort the 'freq_arr[]'
bool compare(pair<int, int> p1, pair<int, int> p2)
{
    // if frequencies of two elements are same
    // then the larger number should come first
    if (p1.second == p2.second)
        return p1.first > p2.first;

    // sort on the basis of decreasing order
    // of frequencies
    return p1.second > p2.second;
}
```

```
    // unordered_map 'um' implemented as frequency hash table
    unordered_map<int, int> um;
    for (int i = 0; i < n; i++)
        um[arr[i]]++;

    // store the elements of 'um' in the vector 'freq_arr'
    vector<pair<int, int> > freq_arr(um.begin(), um.end());

    // sort the vector 'freq_arr' on the basis of the
    // 'compare' function
    sort(freq_arr.begin(), freq_arr.end(), compare);

    // display the top k numbers
    cout << k << " numbers with most occurrences are:\n";
    for (int i = 0; i < k; i++)
        cout << freq_arr[i].first << " ";
}

// Driver program to test above
int main()
{
    int arr[] = { 3, 1, 4, 4, 5, 2, 6, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    print_N_mostFrequentNumber(arr, n, k);
    return 0;
}
```

Java

```
// Java implementation to find
// k elements with max occurrence.
import java.util.*;
public class KFrequentNumbers
{
    static void print_N_mostFrequentNumber(int[] arr,
                                           int n,
                                           int k)
    {
        // distinct elements in Map
        // with element as the key &
        // count as the value.
        for (int i = 0; i < n; i++) {

            // Get the count for the
            // element if already present in the
            // Map or get the default value which is 0.
            mp.put(arr[i],
                   mp.getOrDefault(arr[i], 0) + 1);
        }

        // Create a list from elements of HashMap
        List<Map.Entry<Integer, Integer> > list
            = new ArrayList<Map.Entry<Integer,
                                      Integer> >(
                mp.entrySet());

        // Sort the list
        Collections.sort(
            list,
            new Comparator<Map.Entry<Integer,
                                     Integer> >()
          {
                public int compare(
                    Map.Entry<Integer, Integer> o1,
                    Map.Entry<Integer, Integer> o2)
                {
                    if (o1.getValue() == o2.getValue())
                        return o2.getKey() - o1.getKey();
                    else
                        return o2.getValue()
                            - o1.getValue();
                }
            });

        for (int i = 0; i < k; i++)
            System.out.println(list.get(i).getKey());
    }

    // Driver Code
```

```
        int k = 2;

        // Function call
        print_N_mostFrequentNumber(arr, n, k);
    }
}
```

Python3

```
# Python3 implementation to find k numbers
# with most occurrences in the given array

# function to print the k numbers with
# most occurrences


def pr_N_mostFrequentNumber(arr, n, k):

    um = {}
    for i in range(n):
        if arr[i] in um:
            um[arr[i]] += 1
        else:
            um[arr[i]] = 1
    a = [0] * (len(um))
    j = 0
    for i in um:
        a[j] = [i, um[i]]
        j += 1
    a = sorted(a, key=lambda x: x[0],
               reverse=True)
    a = sorted(a, key=lambda x: x[1],
               reverse=True)

    # display the top k numbers
    print(k, "numbers with most occurrences are:")
    for i in range(k):
        print(a[i][0], end=" ")
    k = 2
    pr_N_mostFrequentNumber(arr, n, k)

# This code is contributed by
# Shubham Singh(SHUBHAMSINGH10)
```

C#

```
// C# implementation to find
// k elements with max occurrence.

using System;
using System.Collections.Generic;

public class Comparer : IComparer<KeyValuePair<int, int> > {
    public int Compare(KeyValuePair<int, int> p2,
                       KeyValuePair<int, int> p1)
    {
        // if frequencies of two elements are same
        // then the larger number should come first
        if (p1.Value == p2.Value)
            return p1.Key.CompareTo(p2.Key);

        // sort on the basis of decreasing order
        // of frequencies
        return p1.Value.CompareTo(p2.Value);
    }
}

public class KFrequentNumbers {
    static void print_N_mostFrequentNumber(int[] arr, int n,
                                           int k)
    {

        IDictionary<int, int> um
            = new Dictionary<int, int>();

        // Put count of all the
        // distinct elements in Map
```

```
            // element if already present in the
            // Map or get the default value which is 0
            if (um.ContainsKey(arr[i]))
                um[arr[i]] += 1;
            else
                um[arr[i]] = 1;
        }

        // Create a list from elements of HashMap
        List<KeyValuePair<int, int> > list
            = new List<KeyValuePair<int, int> >();
        foreach(KeyValuePair<int, int> entry in um)
        {
            list.Add(entry);
        }

        // Sort the list
        Comparer compare = new Comparer();
        list.Sort(compare);

        for (int i = 0; i < k; i++)
            Console.Write(list[i].Key + " ");
    }

    public static void Main(string[] args)
    {
        int[] arr = { 3, 1, 4, 4, 5, 2, 6, 1 };
        int n = arr.Length;
        int k = 2;

        Console.Write(
            k + " elements with most occurrences are:\n");
        // Function call
        print_N_mostFrequentNumber(arr, n, k);
    }
}

// this code is contributed by phasing17
```

Javascript

```
// JavaScript implementation to find
// k elements with max occurrence.

function print_N_mostFrequentNumber(arr, n, k) {

    let mp = new Map();

    // Put count of all the
    // distinct elements in Map
    // with element as the key &
    // count as the value.
    for (let i = 0; i < n; i++) {

        // Get the count for the
        // element if already present in the
        // Map or get the default value which is 0.

        if (mp.has(arr[i])) {
            mp.set(arr[i], mp.get(arr[i]) + 1)
        } else {
            mp.set(arr[i], 1)
        }
    }

    // Create a list from elements of HashMap
    let list = [...mp];

    // Sort the list
    list.sort((o1, o2) => {
        if (o1[1] == o2[1])
            return o2[0] - o1[0];
        else
            return o2[1] - o1[1];
    })

    document.write(k + " numbers with most occurrences are:<br>");
    for (let i = 0; i < k; i++)
        document.write(list[i][0] + " ");
}
```

```
let k = 2;

// Function call
print_N_mostFrequentNumber(arr, n, k); 1

</script>
```

Output

```
2 numbers with most occurrences are:
4 1
```

Complexity Analysis:

- **Time Complexity:** O(d log d), where **d** is the count of distinct elements in the array. To sort the array O(d log d) time is needed.
- **Auxiliary Space:** O(d), where **d** is the count of distinct elements in the array. To store the elements in HashMap O(d) space complexity is needed.

Method 2:

- **Approach:** Create a HashMap to store element-frequency pair in the HashMap. HashMap is used to perform insertion and updation in constant time. Then use a priority queue to store the element-frequency pair (Max-Heap). This gives the element which has maximum frequency at the root of the Priority Queue. Remove the top or root of Priority Queue K times and print the element. To insert and delete the top of the priority queue *O(log n)* time is required.
  Priority Queue: Priority queues are a type of container adapters, specifically designed such that the first element of the queue is the greatest of all elements in the queue and elements are in non increasing order(hence we can see that each element of the queue

1. Create a Hashmap *hm*, to store key-value pair, i.e. element-frequency pair.
2. Traverse the array from start to end.
3. For every element in the array update *hm[array[i]]++*
4. Store the element-frequency pair in a Priority Queue and create the Priority Queue, this takes O(n) time.
5. Run a loop k times, and in each iteration remove the top of the priority queue and print the element.

Below is the Implementation of the above algorithm:

C++

```
// C++ implementation to find k numbers with most
// occurrences in the given array
#include <bits/stdc++.h>

using namespace std;

// comparison function defined for the priority queue
struct compare {
    bool operator()(pair<int, int> p1, pair<int, int> p2)
    {
        // if frequencies of two elements are same
        // then the larger number should come first
        if (p1.second == p2.second)
            return p1.first < p2.first;

        // insert elements in the priority queue on the basis of
        // decreasing order of frequencies
        return p1.second < p2.second;
    }
};

// function to print the k numbers with most occurrences
void print_N_mostFrequentNumber(int arr[], int n, int k)
```

```
        um[arr[i]]++;



    // priority queue 'pq' implemented as max heap on the basis
    // of the comparison operator 'compare'
    // element with the highest frequency is the root of 'pq'
    // in case of conflicts, larger element is the root
    priority_queue<pair<int, int>, vector<pair<int, int> >,
                   compare>
        pq(um.begin(), um.end());

    // display the top k numbers
    cout << k << " numbers with most occurrences are:\n";
    for (int i = 1; i <= k; i++) {
        cout << pq.top().first << " ";
        pq.pop();
    }
}

// Driver program to test above
int main()
{
    int arr[] = { 3, 1, 4, 4, 5, 2, 6, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    print_N_mostFrequentNumber(arr, n, k);
    return 0;
}
```

Java

```
// Java implementation to find k
// elements with max occurrence.
import java.util.*;
public class KFrequentNumbers {
    static void print_N_mostFrequentNumber(int[] arr,
                                           int n,
                                           int k)
```

```
        // distinct elements in Map
        // with element as the key &
        // count as the value.
        for (int i = 0; i < n; i++) {

            // Get the count for the
            // element if already
            // present in the Map or
            // get the default value
            // which is 0.
            mp.put(arr[i],
                   mp.getOrDefault(arr[i], 0) + 1);
        }

        // Create a Priority Queue
        // to sort based on the
        // count or on the key if the
        // count is same
        PriorityQueue<Map.Entry<Integer,
                                Integer>> queue
            = new PriorityQueue<>(
                (a, b)
                -> a.getValue().equals(b.getValue())
                   ? Integer.compare(b.getKey(),
                                     a.getKey())
                   : Integer.compare(b.getValue(),
                                     a.getValue()));

        // Insert the data from the map
        // to the Priority Queue.
        for (Map.Entry<Integer, Integer> entry :
             mp.entrySet())
            queue.offer(entry);

        // Print the top k elements
        for (int i = 0; i < k; i++)
        {
            System.out.println(queue.poll().getKey());
        }
    }
```

```
        int n = arr.length;
        int k = 2;

        // Function call
        print_N_mostFrequentNumber(arr, n, k);
    }
}

// This code is contributed by Shubham Kumar Shah
```

Python3

```
# Python3 implementation to find k
# numbers with most occurrences in
# the given array
import heapq

# Function to print the k numbers with
# most occurrences
def print_N_mostFrequentNumber(arr, n, k):

    mp = dict()

    # Put count of all the distinct elements
    # in dictionary with element as the
    # key & count as the value.
    for i in range(0, n):
        if arr[i] not in mp:
            mp[arr[i]] = 0
        else:
            mp[arr[i]] += 1

    # Using heapq data structure
    heap = [(value, key) for key,
             value in mp.items()]

    # Get the top k elements
    largest = heapq.nlargest(k, heap)
    # Print the top k elements
    for i in range(k):
        print(largest[i][1], end =" ")

# Driver code
if __name__=="__main__":

    arr = [ 3, 1, 4, 4, 5, 2, 6, 1 ]
    n = len(arr)
    k = 2

    print_N_mostFrequentNumber(arr, n, k)

# This code is contributed by MuskanKalra1
```

C#

```
// C# implementation to find k
// elements with max occurrence.
using System;
using System.Collections.Generic;
using System.Linq;

public class KFrequentNumbers {
  static void print_N_mostFrequentNumber(int[] arr, int n,
                                         int k)
  {
    Dictionary<int, int> mp
      = new Dictionary<int, int>();

    // Put count of all the
    // distinct elements in Map
    // with element as the key &
    // count as the value.
    for (int i = 0; i < n; i++) {

      // Get the count for the
      // element if already
      // present in the Map or
```

```
        mp[arr[i]]++;
      }

      // Create a Priority Queue
      // to sort based on the
      // count or on the key if the
      // count is same
      List<int> queue = mp.Keys.ToList();
      queue.Sort(delegate(int y, int x) {
        if (mp[x] == mp[y])
          return x.CompareTo(y);
        else
          return (mp[x]).CompareTo(mp[y]);
      });

      // Print the top k elements
      Console.WriteLine(
        k + " numbers with the most occurrences are:");
      for (int i = 0; i < k; i++) {
        Console.WriteLine(queue[i] + " ");
      }
    }

    // Driver Code
    public static void Main(string[] args)
    {
      int[] arr = { 3, 1, 4, 4, 5, 2, 6, 1 };
      int n = arr.Length;
      int k = 2;

      // Function call
      print_N_mostFrequentNumber(arr, n, k);
    }
  }

// This code is contributed by phasing17
```

Javascript

```
<script>
```

```
    {
        let mp=new Map();
        // Put count of all the
            // distinct elements in Map
            // with element as the key &
            // count as the value.
            for (let i = 0; i < n; i++) {

                // Get the count for the
                // element if already
                // present in the Map or
                // get the default value
                // which is 0.
                if(!mp.has(arr[i]))
                    mp.set(arr[i],0);

                mp.set(arr[i],
                       mp.get(arr[i]) + 1);
            }

            // Create a Priority Queue
            // to sort based on the
            // count or on the key if the
            // count is same
            let queue=[...mp];

            queue.sort(function(a,b){
                if(a[1]==b[1])
                {
                    return b[0]-a[0];
                }
                else
                {
                    return b[1]-a[1];
                }
            });

            document.write(k+" numbers with most "+"occurrences are:<br>")
            for(let i=0;i<k;i++)
            {
                document.write(queue[i][0]+" ");
```

```
let n = arr.length;
let k = 2;

// Function call
print_N_mostFrequentNumber(arr, n, k);

// This code is contributed by avanitrachhadiya2155
</script>
```

Output

```
2 numbers with most occurrences are:
4 1
```

Complexity Analysis:

- **Time Complexity:** O(k log d + d), where **d** is the count of distinct elements in the array.
  To remove the top of priority queue O(log d) time is required, so if k elements are removed then O(k log d) time is required and to traverse the distinct elements O(d) time is required.
- **Auxiliary Space:** O(d), where **d** is the count of distinct elements in the array.
  To store the elements in HashMap O(d) space complexity is needed.

Find k most frequent in linear time

This article is contributed by **Ayush Jauhari**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Start Your Coding Journey Now! Login Register

Like 56

Previous

k-th smallest absolute difference of two elements in an array

Next

Maximum sum such that no two elements are adjacent

RECOMMENDED ARTICLES

Page : 1 2 3

01 Find given occurrences of Mth most frequent element of Array
21, Apr 22

02 Minimize splits in given Array to find subsets of at most 2 elements with sum at most K
26, Dec 21

03 Minimum operations to make a numeric string palindrome by removing at most 2 unique character occurrences

05 Find numbers starting from 1 with sum at-most K excluding given numbers
02, Dec 21

06 Find the two numbers with odd occurrences in an unsorted array
24, May 12

07 Minimum possible value T such that at most D Partitions of the Array having at most sum T is possible

occurrences of a most frequent element
17, Dec 17

and difference between adjacent elements at most 1
22, Jan 21

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Improved By : SHUBHAMSINGH10, nidhi_biet, Akanksha_Rai, andrew1234, shubhamshah92, rohitkumar52, MuskanKalra1, kshitiz13, _saurabh_jaiswal, avanitrachhadiya2155, anikaseth98, ruhelaa48, phasing17

Article Tags : Amazon, Order-Statistics, STL, Arrays, Hash, Heap

Practice Tags : Amazon, Arrays, Hash, Heap, STL

Improve Article Report Issue

Load Comments

NodeJS