Select a random number from stream, with O(1) space

Given a stream of numbers, generate a random number from the stream. You are allowed to use only O(1) space and the input is in the form of stream, so can't store the previously seen numbers.

So how do we generate a random number from the whole stream such that the probability of picking any number is 1/n. with O(1) extra space? This problem is a variation of Reservoir Sampling. Here the value of k is 1.

1) Initialize 'count' as 0, 'count' is used to store count of numbers seen so far in stream.
2) For each number 'x' from stream, do following
…..**a)** Increment 'count' by 1.
…..**b)** If count is 1, set result as x, and return result.
…..**c)** Generate a random number from 0 to 'count-1'. Let the generated random number be i.
…..**d)** If i is equal to 'count – 1', update the result as x.

C/C++

```
// An efficient C program to randomly select a number from stream of numbers.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A function to randomly select a item from stream[0], stream[1], .. stream[i-1]
int selectRandom(int x)
{
    static int res;    // The resultant random number
    static int count = 0;  //Count of numbers visited so far in stream

    count++;  // increment count of numbers seen so far

    // If this is the first element from stream, return it
    if (count == 1)
        res = x;
    else
    {
        // Generate a random number from 0 to count - 1
        int i = rand() % count;

        // Replace the prev random number with new number with 1/count probability
```

```
        if (i == count - 1)
            res  = x;
    }
    return res;
}

// Driver program to test above function.
int main()
{
    int stream[] = {1, 2, 3, 4};
    int n = sizeof(stream)/sizeof(stream[0]);

    // Use a different seed value for every run.
    srand(time(NULL));
    for (int i = 0; i < n; ++i)
        printf("Random number from first %d numbers is %d \n",
                                  i+1, selectRandom(stream[i]));
    return 0;
}
```

Run on IDE

Java

```
//An efficient Java program to randomly select a number from stream of numbers.

import java.util.Random;

public class GFG
{
    static int res = 0;    // The resultant random number
    static int count = 0;  //Count of numbers visited so far in stream

    //A method to randomly select a item from stream[0], stream[1], .. stream[i-1]
    static int selectRandom(int x)
    {
        count++; // increment count of numbers seen so far

        // If this is the first element from stream, return it
        if (count == 1)
            res = x;
        else
        {
             // Generate a random number from 0 to count - 1
            Random r = new Random();
            int i = r.nextInt(count);

            // Replace the prev random number with new number with 1/count probability
            if(i == count - 1)
                res = x;
        }
        return res;
    }

    // Driver program to test above function.
    public static void main(String[] args)
    {
        int stream[] = {1, 2, 3, 4};
        int n = stream.length;
        for(int i = 0; i < n; i++)
            System.out.println("Random number from first " + (i+1) +
                               " numbers is " + selectRandom(stream[i]));
    }
}
//This code is contributed by Sumit Ghosh
```

Run on IDE

Output:

```
Random number from first 1 numbers is 1
Random number from first 2 numbers is 1
Random number from first 3 numbers is 3
Random number from first 4 numbers is 4
```

Auxiliary Space: O(1)

How does this work

We need to prove that every element is picked with 1/n probability where n is the number of items seen so far. For every new stream item x, we pick a random number from 0 to 'count -1', if the picked number is 'count-1', we replace the previous result with x.

To simplify proof, let us first consider the last element, the last element replaces the previously stored result with 1/n probability. So probability of getting last element as result is 1/n.

Let us now talk about second last element. When second last element processed first time, the probability that it replaced the previous result is 1/(n-1). The probability that previous result stays when nth item is considered is (n-1)/n. So probability that the second last element is picked in last iteration is [1/(n-1)] * [(n-1)/n] which is 1/n.

Similarly, we can prove for third last element and others.

References:
Reservoir Sampling

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Mathematical Randomized array-stream Random Algorithms

Login to Improve this Article

Recommended Posts:

Reservoir Sampling
Find the largest multiple of 2, 3 and 5
Random number generator in arbitrary probability distribution fashion
Shuffle a given array

Stream toArray() in Java with Examples

Stream toArray() returns an array containing the elements of this stream. It is a ***terminal operation*** i.e, it may traverse the stream to produce a result or a side-effect. After the terminal operation is performed, the stream pipeline is considered consumed, and can no longer be used.

Syntax :

```
Object[] toArray()
```

Return Value : The function returns an array containing the elements of this stream.

Example 1 :

```
// Java code for Stream toArray()
import java.util.*;
import java.util.stream.Stream;

class GFG {

    // Driver code
    public static void main(String[] args)
    {
        // Creating a Stream of Integers
        Stream<Integer> stream = Stream.of(5, 6, 7, 8, 9, 10);

        // Using Stream toArray()
        Object[] arr = stream.toArray();

        // Displaying the elements in array arr
        System.out.println(Arrays.toString(arr));
    }
}
```

Run on IDE

Output :

```
[5, 6, 7, 8, 9, 10]
```

Example 2 :

```
// Java code for Stream toArray()
import java.util.*;
import java.util.stream.Stream;

class GFG {

    // Driver code
    public static void main(String[] args)
    {
        // Creating a Stream of Strings
        Stream<String> stream = Stream.of("Geeks", "for",
                                          "Geeks", "GeeksQuiz");

        // Using Stream toArray()
        Object[] arr = stream.toArray();

        // Displaying the elements in array arr
        System.out.println(Arrays.toString(arr));
    }
}
```

Run on IDE

Output :

```
[Geeks, for, Geeks, GeeksQuiz]
```

Example 3 :

```
// Java code for Stream toArray()
import java.util.*;
import java.util.stream.Stream;

class GFG {

    // Driver code
    public static void main(String[] args)
    {
        // Creating a Stream of Strings
        Stream<String> stream = Stream.of("Geeks", "for",
                                          "gfg", "GeeksQuiz");

        // Using Stream toArray() and filtering
        // the elements that starts with 'G'
        Object[] arr = stream.filter(str
                                     -> str.startsWith("G"))
                           .toArray();

        // Displaying the elements in array arr
        System.out.println(Arrays.toString(arr));
    }
}
```

Run on IDE

Output :

```
[Geeks, GeeksQuiz]
```

Streams on Arrays in Java 8

In this article, we would be going through stream method of Arrays class which is added in Java 8, it simplifies many operations on arrays as well have improved the efficiency.
Addition of different features like lambdas and streams in java 8 have made java efficient to write elegant code which have improve the readability providing increase in efficiency of performance in most case .

Syntax :

```
public static IntStream stream(int[] arr)
Parameter :
arr - An array which is to be converted to the stream
Returns :
An IntStream of an array
```

Variations :

```
public static IntStream stream(int[] array)
public static IntStream stream(int[] array, int startInclusive, int endExclusive)
public static DoubleStream stream(double[] array)

public static DoubleStream stream(double[] array, int startInclusive, int endExclusive)
public static LongStream stream(long[] array)
public static LongStream stream(long[] array, int startInclusive, int endExclusive)
public static  Stream stream(T[] array)
public static  Stream stream(T[] array, int startInclusive, int endExclusive)
```

Prerequisite :-

- Lambda Expressions in Java 8
- Stream In Java

Note: – Even if you are not familiar with these topics, you can go through the article as it uses very basic lambda expression and explains how to use method of stream class.
Let's see an example of streams on Arrays. In this example we will be finding average over the array

elements and will see the difference in way of writing code in imperative and declarative styles

Example 1:

```
import java.util.Arrays;
class GFG_Demo_1 {
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
            11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

        // Let's try the imperative style first(which we
        // are familiar with)
        int sum = 0;
        for (int i = 0; i < arr.length; i++)
            sum += arr[i];
        System.out.println("Average using iteration :" +
                                    (sum / arr.length));

        // Let's try the declarative style now
        sum = Arrays.stream(arr) // Step 1
                  .sum(); // Step 2
        System.out.println("Average using streams : " +
                                   (sum / arr.length));

        // forEach()
        // It iterates through the entire streams
        System.out.println("Printing array elements : ");
        Arrays.stream(arr)
            .forEach(e->System.out.print(e + " "));
    }
}
```

Run on IDE

Output:

```
Average using iteration :10
Average using streams : 10
Printing array elements :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

In the above example you have seen stream working let's see what these step does.

Step 1:

Arrays.stream(arr) – In this step we call the stream method on the Arrays class passing arr as the parameter to the function this statement returns IntStream.

Step 2:

Arrays.stream(arr).sum() – Once we get the IntStream we can use different methods of the IntStream interface.

While you go through IntStream. interface documentation you can open each method to see whether it's perform a terminal operation or intermediate operation. And we should use this method accordingly either at the terminal or in between.

Now let's go through different methods of IntStream and see what operations this methods perform.We will

see example of all this methods in contrast of an array..

We will going through the following methods in the example below.

1. asDoubleStream()
2. asLongStream()
3. anyMatch()
4. allMatch()
5. noneMatch()

```
import java.util.Arrays;
import java.util.function.IntPredicate;
class GFG_Demo_2 {
  public static void main(String[] args)
  {
        int arr_sample1[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                  11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

        // asDoubleStream()
        // It converts the original array to double
        System.out.println("Example of asDoubleStream(): ");
        Arrays.stream(arr_sample1)
            .asDoubleStream()
            .forEach(e->System.out.print(e + " "));

        // asLongStream()
        // It converts the original array to Long
        System.out.println("\nExample of asLongStream");
        Arrays.stream(arr_sample1)
            .asLongStream()
            .forEach(e->System.out.print(e + " "));

        int arr_sample2[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9,
            10, 23, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

        // anyMatch()
        // This method find whether the given predicate
        // is in the array or not
        System.out.println("\nExample of anyMatch");

        // Test whether any of the element in array is
        // divisible by 11 or not
        IntPredicate predicate = e->e % 11 == 0;
        System.out.println(Arrays.stream(arr_sample2)
                              .anyMatch(predicate));

        // You can directly write the lambda expression
        // which computes to IntPredicate
        // Uncomment to test
        // System.out.println(Arrays.stream(arr)
        // .anyMatch(e -> e % 11 == 0));

        int arr_sample3[] = { 2, 4, 6, 8, 10 };
        int arr_sample4[] = { 1, 3, 5, 7, 11 };

        // allMatch()
        // This method finds whether the given predicate
        // matches the entire array or not
        System.out.println("Example of allMatch :");

        // Returns true as all the elements of arr_sample3
        // is even
        System.out.println(Arrays.stream(arr_sample3)
                              .allMatch(e->e % 2 == 0));
```

```
        // Returns false as all the elements of arr_sammple4
        // is odd
        System.out.println(Arrays.stream(arr_sample4)
                                 .allMatch(e->e % 2 == 0));

        // noneMatch()
        System.out.println("Example of noneMatch");
        System.out.println(Arrays.stream(arr_sample4)
                                 .noneMatch(e->e % 2 == 0));
    }
}
```

Run on IDE

Output:

```
Example of asDoubleStream():
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0
14.0 15.0 16.0 17.0 18.0 19.0 20.0
Example of asLongStream
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Example of anyMatch
false
Example of allMatch :
true
false
Example of noneMatch
true
```

We have seen very few methods though IntStream provides many more, lets try some more.

We will going through the following methods in the example below.

1. average()
2. findAny()
3. findFirst()
4. max()
5. min()
6. reduce()

Remember all this method returns OptionalInt or OptionalDouble instead of int or double.

```
import java.util.Arrays;
class GFG_Demo_3 {
    public static void main(String[] args)
    {
        int arr_sample1[] = { 11, 2, 3, 42, 5, 6, 17, 8, 9,
               10, 11, 12, 13, 24, 55, 16, 47, 18, 19, 20 };
        System.out.println("These method retuns Optional");

        // average()
        // This method returns a average of an array
        System.out.println("Example of average() : ");
        System.out.println((Arrays.stream(arr_sample1)
                                  .average()));

        // findAny()
```

```
        // It can return any value from the stream
        // Most of the time it returns the first value
        // but it is not assured it can return any value
        System.out.println("Example of findAny() : ");
        System.out.println(Arrays.stream(arr_sample1)
                                 .findAny());

        // findFirst()
        // It returns the first element of the stream
        System.out.println("Example of findFirst() :");
        System.out.println(Arrays.stream(arr_sample1)
                                 .findAny());

        // max()
        // It returns the max element in an array
        System.out.println("Example of max() :");
        System.out.println(Arrays.stream(arr_sample1)
                                 .max());

        // min()
        // It returns the max element in an array
        System.out.println("Example of max() :");
        System.out.println(Arrays.stream(arr_sample1)
                                 .min());

        // reduce()
        // It reduces the array by certain operation
        // Here it performs addition of array elements
        System.out.println("Example of reduce() :");
        System.out.println(Arrays.stream(arr_sample1)
                                 .reduce((x, y)->x + y));

        // reduce() have another variation which we will
        // see in different example
    }
}
```

Run on IDE

Output:

```
These method retuns Optional
Example of average() :
OptionalDouble[17.4]
Example of findAny() :
OptionalInt[11]
Example of findFirst() :
OptionalInt[11]
Example of max() :
OptionalInt[55]
Example of max() :
OptionalInt[2]
Example of reduce() :
OptionalInt[348]
```

But it becomes really difficult to work with this OptionalInt and OptionalDouble, hence Java provides method to convert them into double and int values such that it can be easily reused

```
import java.util.Arrays;
class GFG_Demo_4 {
public static void main(String[] args)
    {
```

```
        int arr_sample1[] = { 11, 2, 3, 42, 5, 6, 17, 8, 9,
                10, 11, 12, 13, 24, 55, 16, 47, 18, 19, 20 };
        System.out.println("These method convert Optional to primitive");

        // OptionalDouble can be converted to double by using getAsDouble()
        // if average doesn't contains any value it throws NoSuchElementException
        System.out.println("Example of average() : ");
        System.out.println((Arrays.stream(arr_sample1)
                                  .average()
                                  .getAsDouble()));

        // OptionalInt can be converted to int by using getAsInt()
        System.out.println("Example of findAny() : ");
        System.out.println(Arrays.stream(arr_sample1)
                                 .findAny()
                                 .getAsInt());
    }
}
```

Run on IDE

Output:

```
These method convert Optional to primitive
Example of average() :
17.4
Example of findAny() :
11
```

There are some more methods provided by IntStream which we will be going through in different article and would be working with different variations of stream method.

Reference :

https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

https://docs.oracle.com/javase/8/docs/api/java/util/stream/IntStream.html

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

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Java array-stream java-stream

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Average of a stream of numbers

Difficulty Level: Rookie

Given a stream of numbers, print average (or mean) of the stream at every point. For example, let us consider the stream as 10, 20, 30, 40, 50, 60, …

```
Average of 1 numbers is 10.00
Average of 2 numbers is 15.00
Average of 3 numbers is 20.00
Average of 4 numbers is 25.00
Average of 5 numbers is 30.00
Average of 6 numbers is 35.00
.................
```

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

To print mean of a stream, we need to find out how to find average when a new number is being added to the stream. To do this, all we need is count of numbers seen so far in the stream, previous average and new number. Let *n* be the count, *prev_avg* be the previous average and x be the new number being added. The average after including *x* number can be written as *(prev_avg*n + x)/(n+1)*.

C++

```
#include <stdio.h>

// Returns the new average after including x
float getAvg(float prev_avg, int x, int n)
{
    return (prev_avg * n + x) / (n + 1);
}

// Prints average of a stream of numbers
```

```
void streamAvg(float arr[], int n)
{
    float avg = 0;
    for (int i = 0; i < n; i++) {
        avg = getAvg(avg, arr[i], i);
        printf("Average of %d numbers is %f \n", i + 1, avg);
    }
    return;
}

// Driver program to test above functions
int main()
{
    float arr[] = { 10, 20, 30, 40, 50, 60 };
    int n = sizeof(arr) / sizeof(arr[0]);
    streamAvg(arr, n);

    return 0;
}
```

Run on IDE

Java

```
// Java program to find average
// of a stream of numbers
class GFG {

    // Returns the new average after including x
    static float getAvg(float prev_avg, float x, int n)
    {
        return (prev_avg * n + x) / (n + 1);
    }

    // Prints average of a stream of numbers
    static void streamAvg(float arr[], int n)
    {
        float avg = 0;
        for (int i = 0; i < n; i++)
        {
            avg = getAvg(avg, arr[i], i);
            System.out.printf("Average of %d numbers is %f \n",
                                                    i + 1, avg);
        }
        return;
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        float arr[] = { 10, 20, 30, 40, 50, 60 };
        int n = arr.length;
        streamAvg(arr, n);
    }
}

// This code is contributed by Smitha Dinesh Semwal
```

Run on IDE

Output :

```
Average of 1 numbers is 10.000000
Average of 2 numbers is 15.000000
Average of 3 numbers is 20.000000
Average of 4 numbers is 25.000000
Average of 5 numbers is 30.000000
Average of 6 numbers is 35.000000
```

The above function getAvg() can be optimized using following changes. We can avoid the use of prev_avg and number of elements by using static variables (Assuming that only this function is called for average of stream). Following is the oprimnized version.

```
#include <stdio.h>

// Returns the new average after including x
float getAvg(int x)
{
    static int sum, n;

    sum += x;
    return (((float)sum) / ++n);
}

// Prints average of a stream of numbers
void streamAvg(float arr[], int n)
{
    float avg = 0;
    for (int i = 0; i < n; i++) {
        avg = getAvg(arr[i]);
        printf("Average of %d numbers is %f n", i + 1, avg);
    }
    return;
}

// Driver program to test above functions
int main()
{
    float arr[] = { 10, 20, 30, 40, 50, 60 };
    int n = sizeof(arr) / sizeof(arr[0]);
    streamAvg(arr, n);

    return 0;
}
```

Run on IDE

Thanks to Abhijeet Deshpande for suggesting this optimized version.

Related article:

Program for average of an array (Iterative and Recursive)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Mathematical array-stream

Login to Improve this Article

Check divisibility in a binary stream

Stream of binary number is coming, the task is to tell the number formed so far is divisible by a given number n.

At any given time, you will get 0 or 1 and tell whether the number formed with these bits is divisible by n or not.

Generally, e-commerce companies ask this type of questions. It was asked me in Microsoft interview. Actually that question was a bit simple, interviewer fixed the n to 3.

Method 1 (Simple but causes overflow):

Keep on calculating the number formed and just check divisibility by n.

```
/* Simple implementation of the logic,
   without error handling compiled with
   Microsoft visual studio 2015 */
void CheckDivisibility2(int n)
{
    int num = 0;
    std::cout << "press any key other than"
                 " 0 and 1 to terminate \n";
    while (true)
    {
        int incomingBit;

        // read next incoming bit via standard
        // input. 0, 00, 000.. are same as int 0
        // ans 1, 01, 001, 00..001 is same as 1.
        std::cin >> incomingBit;

        // Update value of num formed so far
        if (incomingBit == 1)
            num = (num * 2 + 1);
        else if (incomingBit == 0)
            num = (num * 2);

        else
            break;
        if (num % n == 0)
            std::cout << "yes \n";
        else
            std::cout << "no \n";
    }
}
```

Run on IDE

Problem in this solution: What about the overflow. Since 0 and 1 will keep on coming and the number formed will go out of range of integer.

Method 2 (Doesn't cause overflow) :

In this solution, we just maintain the remainder if remainder is 0, the formed number is divisible by n otherwise not. This is the same technique that is used in Automata to remember the state. Here also we are remembering the state of divisibility of input number.

In order to implement this technique, we need to observe how the value of a binary number changes, when it is appended by 0 or 1.

Let's take an example. Suppose you have binary number 1.
If it is appended by 0 it will become 10 (2 in decimal) means 2 times of the previous value.
If it is appended by 1 it will become 11(3 in decimal), 2 times of previous value +1.

How does it help in getting the remainder?
Any number (n) can be written in the form m = an + r where a, n and r are integers and r is the remainder. So when m is multiplied by any number so the remainder. Suppose m is multiplied by x so m will be mx = xan + xr. so (mx)%n = (xan)%n + (xr)%n = 0 + (xr)%n = (xr)%n;

We need to just do the above calculation (calculation of value of number when it is appended by 0 or 1) only over remainder.

```
When a binary number is appended by 0 (means
multiplied by 2), the new remainder can be
calculated based on current remainder only.
r = 2*r % n;

And when a binary number is appended by 1.
r = (2*r + 1) % n;
```

```
// C++ program to check divisibility in a stream
#include <iostream>
using namespace std;

/*  A very simple implementation of the logic,
    without error handling. just to demonstrate
    the above theory. This simple version not
    restricting user from typing 000, 00 , 000.. ,
    because this all will be same as 0 for integer
    same is true for 1, 01, 001, 000...001 is same
    as 1, so ignore this type of error handling
    while reading just see the main logic is correct. */
void CheckDivisibility(int n)
{
    int remainder = 0;
```

```
    std::cout << "press any key other than 0"
                 " and 1 to terminate \n";
    while (true)
    {
        // Read next incoming bit via standard
        // input. 0, 00, 000.. are same as int 0
        // ans 1, 01, 001, 00..001 is same as 1.
        int incomingBit;
        cin >> incomingBit;

        // Update remainder
        if (incomingBit == 1)
            remainder = (remainder * 2 + 1) % n;
        else if (incomingBit == 0)
            remainder = (remainder * 2) % n;
        else
            break;

        // If remainder is 0.
        if (remainder % n == 0)
            cout << "yes \n";
        else
            cout << "no \n";
    }
}

// Driver code
int main()
{
    CheckDivisibility(3);
    return 0;
}
```

Run on IDE

Input:

```
1
0
1
0
1
-1
```

Output:

```
Press any key other than 0 and 1 to terminate
no
no
no
no
yes
```

Related Articles:

DFA based division

Check if a stream is Multiple of 3

Find the first non-repeating character from a stream of characters

Given a stream of characters, find the first non-repeating character from stream. You need to tell the first non-repeating character in O(1) time at any moment.

If we follow the first approach discussed here, then we need to store the stream so that we can traverse it one more time to find the first non-repeating character at any moment. If we use extended approach discussed in the same post, we need to go through the count array every time first non-repeating element is queried. We can find the first non-repeating character from stream at any moment without traversing any array.

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

The idea is to use a DLL (**D**oubly **L**inked **L**ist) to efficiently get the first non-repeating character from a stream. The DLL contains all non-repeating characters in order, i.e., the head of DLL contains first non-repeating character, the second node contains the second non-repeating and so on.
We also maintain two arrays: one array is to maintain characters that are already visited two or more times, we call it repeated[], the other array is array of pointers to linked list nodes, we call it inDLL[]. The size of both arrays is equal to alphabet size which is typically 256.

1. Create an empty DLL. Also create two arrays inDLL[] and repeated[] of size 256. inDLL is an array of pointers to DLL nodes. repeated[] is a boolean array, repeated[x] is true if x is repeated two or more times, otherwise false. inDLL[x] contains pointer to a DLL node if character x is present in DLL, otherwise NULL.
2. Initialize all entries of inDLL[] as NULL and repeated[] as false.
3. To get the first non-repeating character, return character at head of DLL.

4. Following are steps to process a new character ‘x’ in stream.
    - If repeated[x] is true, ignore this character (x is already repeated two or more times in the stream)
    - If repeated[x] is false and inDLL[x] is NULL (x is seen first time). Append x to DLL and store address of new DLL node in inDLL[x].
    - If repeated[x] is false and inDLL[x] is not NULL (x is seen second time). Get DLL node of x using inDLL[x] and remove the node. Also, mark inDLL[x] as NULL and repeated[x] as true.

Note that appending a new node to DLL is O(1) operation if we maintain tail pointer. Removing a node from DLL is also O(1). So both operations, addition of new character and finding first non-repeating character take O(1) time.

C/C++

```
// A C++ program to find first non-repeating character
// from a stream of characters
#include <iostream>
#define MAX_CHAR 256
using namespace std;

// A linked list node
struct node
{
    char a;
    struct node *next, *prev;
};

// A utility function to append a character x at the end
// of DLL. Note that the function may change head and tail
// pointers, that is why pointers to these pointers are passed.
void appendNode(struct node **head_ref, struct node **tail_ref,
                char x)
{
    struct node *temp = new node;
    temp->a = x;
    temp->prev = temp->next = NULL;

    if (*head_ref == NULL)
    {
        *head_ref = *tail_ref = temp;
        return;
    }
    (*tail_ref)->next = temp;
    temp->prev = *tail_ref;
    *tail_ref = temp;
}

// A utility function to remove a node 'temp' fromt DLL.
// Note that the function may change head and tail pointers,
// that is why pointers to these pointers are passed.
void removeNode(struct node **head_ref, struct node **tail_ref,
                struct node *temp)
{
    if (*head_ref == NULL)
        return;

    if (*head_ref == temp)
        *head_ref = (*head_ref)->next;
    if (*tail_ref == temp)
```

```
        *tail_ref = (*tail_ref)->prev;
    if (temp->next != NULL)
        temp->next->prev = temp->prev;
    if (temp->prev != NULL)
        temp->prev->next = temp->next;

    delete(temp);
}

void findFirstNonRepeating()
{
    // inDLL[x] contains pointer to a DLL node if x is present
    // in DLL. If x is not present, then inDLL[x] is NULL
    struct node *inDLL[MAX_CHAR];

    // repeated[x] is true if x is repeated two or more times.
    // If x is not seen so far or x is seen only once. then
    // repeated[x] is false
    bool repeated[MAX_CHAR];

    // Initialize the above two arrays
    struct node *head = NULL, *tail = NULL;
    for (int i = 0; i < MAX_CHAR; i++)
    {
        inDLL[i] = NULL;
        repeated[i] = false;
    }

    // Let us consider following stream and see the process
    char stream[] = "geeksforgeeksandgeeksquizfor";
    for (int i = 0; stream[i]; i++)
    {
        char x = stream[i];
        cout << "Reading " << x << " from stream n";

        // We process this character only if it has not occurred
        // or occurred only once. repeated[x] is true if x is
        // repeated twice or more.s
        if (!repeated[x])
        {
            // If the character is not in DLL, then add this at
            // the end of DLL.
            if (inDLL[x] == NULL)
            {
                appendNode(&head, &tail, stream[i]);
                inDLL[x] = tail;
            }
            else // Otherwise remove this character from DLL
            {
                removeNode(&head, &tail, inDLL[x]);
                inDLL[x] = NULL;
                repeated[x] = true; // Also mark it as repeated
            }
        }

        // Print the current first non-repeating character from
        // stream
        if (head != NULL)
            cout << "First non-repeating character so far is "
                 << head->a << endl;
    }
}

/* Driver program to test above function */
int main()
{
    findFirstNonRepeating();
    return 0;
}
```

Run on IDE

Java

```
//A Java program to find first non-repeating character
//from a stream of characters

import java.util.ArrayList;
import java.util.List;

public class NonReapeatingC
{
    final static int MAX_CHAR = 256;

    static void findFirstNonRepeating()
    {
        // inDLL[x] contains pointer to a DLL node if x is present
        // in DLL. If x is not present, then inDLL[x] is NULL
        List<Character> inDLL =new ArrayList<Character>();

        // repeated[x] is true if x is repeated two or more times.
        // If x is not seen so far or x is seen only once. then
        // repeated[x] is false
        boolean[] repeated =new boolean[MAX_CHAR];

        // Let us consider following stream and see the process
        String stream = "geeksforgeeksandgeeksquizfor";
        for (int i=0;i < stream.length();i++)
        {
            char x = stream.charAt(i);
            System.out.println("Reading "+ x +" from stream n");

            // We process this character only if it has not occurred
            // or occurred only once. repeated[x] is true if x is
            // repeated twice or more.s
            if(!repeated[x])
            {
                // If the character is not in DLL, then add this at
                // the end of DLL.
                if(!(inDLL.contains(x)))
                {
                    inDLL.add(x);
                }
                else    // Otherwise remove this character from DLL
                {
                    inDLL.remove((Character)x);
                    repeated[x] = true; // Also mark it as repeated
                }
            }

            // Print the current first non-repeating character from
            // stream
            if(inDLL.size() != 0)
            {
                System.out.print("First non-repeating character so far is ");
                System.out.println(inDLL.get(0));
            }
        }
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        findFirstNonRepeating();
    }
}
```

Find top k (or most frequent) numbers in a stream

Given an array of n numbers. Your task is to read numbers from the array and keep at-most K numbers at the top (According to their decreasing frequency) every time a new number is read. We basically need to print top k numbers sorted by frequency when input stream has included k distinct elements, else need to print all distinct elements sorted by frequency.

Examples:

```
Input :  arr[] = {5, 2, 1, 3, 2}
             k = 4
Output : 5 2 5 1 2 5 1 2 3 5 2 1 3 5

Input  : arr[] = {5, 2, 1, 3, 4}
             k = 4
Output : 5 2 5 1 2 5 1 2 3 5 1 2 3 4
```

Expected time complexity is O(n * k)

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

Explanation of 1st example:
Given array is arr[] = {5, 2, 1, 3, 2} and k = 4
Step 1:After reading 5, there is only one element 5 whose frequency is max till now. so print 5.
Step 2:After reading 2, we will have two elements 2 and 5 with same frequency. As 2, is smaller than 5 but their frequency is same so we will print 2 5.
Step 3: After reading 1, we will have 3 elements 1,2 and 5 with same frequency, so print 1 2 5.
Step 4: Similarly after reading 3, print 1 2 3 5

Step 5: After reading last element 2, since 2 has already occurred so we have now frequency of 2 as 2. So we keep 2 at the top and then rest of element with same frequency in sorted order. So print, 2 1 3 5.

Below is the step by step algorithm to do this:

1. Iterate through the array which contains stream of numbers.
2. To keep track of top k elements, make a top vector of size k+1.
3. For every element in the stream increase its frequency and store it in the last position of top vector. We can use hashing for efficiently fetching frequency of an element and increasing it.
4. Now find the position of element in top vector and iterate from that position to zero. For finding position we can make use of the find() function in C++ STL, it returns an iterator pointing to element if found in the vector.
5. And make that list of k+1 numbers sorted according to frequency and their value.
6. Print top k elements form top vector.
7. Repeat the above steps for every element in the stream.

Below is the C++ implementation of above idea:

C++

```
// C++ program to find top k elements in a stream
#include <bits/stdc++.h>
using namespace std;

// Function to print top k numbers
void kTop(int a[], int n, int k)
{
    // vector of size k+1 to store elements
    vector<int> top(k + 1);

    // array to keep track of frequency
    unordered_map<int, int> freq;

    // iterate till the end of stream
    for (int m = 0; m < n; m++)
    {
        // increase the frequency
        freq[a[m]]++;

        // store that element in top vector
        top[k] = a[m];

        // search in top vector for same element
        auto it = find(top.begin(), top.end() - 1, a[m]);

        // iterate from the position of element to zero
        for (int i = distance(top.begin(), it) - 1; i >= 0; --i)
        {
            // compare the frequency and swap if higher
            // frequency element is stored next to it
            if (freq[top[i]] < freq[top[i + 1]])
                swap(top[i], top[i + 1]);

            // if frequency is same compare the elements
            // and swap if next element is high
            else if ((freq[top[i]] == freq[top[i + 1]])
```

```
                    && (top[i] > top[i + 1]))
                swap(top[i], top[i + 1]);
            else
                break;
        }

        // print top k elements
        for (int i = 0; i < k && top[i] != 0; ++i)
            cout << top[i] << ' ';
    }
    cout << endl;
}

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

Run on IDE

Python

```
# Python program to find top k elements in a stream

# Function to print top k numbers
def kTop(a, n, k):

    # list of size k+1 to store elements
    top = [0 for i in range(k + 1)]

    # dictionary to keep track of frequency
    freq = {i:0 for i in range(k + 1)}

    # iterate till the end of stream
    for m in range(n):

        # increase the frequency
        if a[m] in freq.keys():
            freq[a[m]] += 1
        else:
            freq[a[m]] = 1

        # store that element in top vector
        top[k] = a[m]

        i = top.index(a[m])
        i -= 1

        while i >= 0:

            # compare the frequency and swap if higher
            # frequency element is stored next to it
            if (freq[top[i]] < freq[top[i + 1]]):
                t = top[i]
                top[i] = top[i + 1]
                top[i + 1] = t

            # if frequency is same compare the elements
            # and swap if next element is high
            elif ((freq[top[i]] == freq[top[i + 1]]) and (top[i] > top[i + 1])):
                t = top[i]
                top[i] = top[i + 1]
```

```
                top[i + 1] = t
            else:
                break
            i -= 1

        # print top k elements
        i = 0
        while i < k and top[i] != 0:
            print top[i],
            i += 1
    print

# Driver program to test above function
k = 4
arr = [ 5, 2, 1, 3, 2 ]
n = len(arr)
kTop(arr, n, k)

# This code is contributed by Sachin Bisht
```

Run on IDE

Output:

```
5 2 5 1 2 5 1 2 3 5 2 1 3 5
```

Time Complexity: O(n * k)

Asked in: Amazon

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

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Arrays Hash array-stream cpp-unordered_map Order-Statistics

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Find four elements a, b, c and d in an array such that a+b = c+d
Find the transition point in a binary array
Minimum Index Sum for Common Elements of Two Lists

K’th largest element in a stream

Given an infinite stream of integers, find the k’th largest element at any point of time.

Example:

```
Input:
stream[] = {10, 20, 11, 70, 50, 40, 100, 5, ...}
k = 3

Output:    {_,   _, 10, 11, 20, 40, 50,  50, ...}
```

Extra space allowed is O(k).

Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.

We have discussed different approaches to find k’th largest element in an array in the following posts.
K’th Smallest/Largest Element in Unsorted Array | Set 1
K’th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)
K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)

Here we have a stream instead of whole array and we are allowed to store only k elements.

A **Simple Solution** is to keep an array of size k. The idea is to keep the array sorted so that the k’th largest element can be found in O(1) time (we just need to return first element of array if array is sorted in increasing order)
How to process a new element of stream?
For every new element in stream, check if the new element is smaller than current k’th largest element. If

yes, then ignore it. If no, then remove the smallest element from array and insert new element in sorted order. Time complexity of processing a new element is O(k).

A **Better Solution** is to use a Self Balancing Binary Search Tree of size k. The k'th largest element can be found in O(Logk) time.
How to process a new element of stream?
For every new element in stream, check if the new element is smaller than current k'th largest element. If yes, then ignore it. If no, then remove the smallest element from the tree and insert new element. Time complexity of processing a new element is O(Logk).

An **Efficient Solution** is to use Min Heap of size k to store k largest elements of stream. The k'th largest element is always at root and can be found in O(1) time.
How to process a new element of stream?
Compare the new element with root of heap. If new element is smaller, then ignore it. Otherwise replace root with new element and call heapify for the root of modified heap. Time complexity of finding the k'th largest element is O(Logk).

```
// A C++ program to find k'th smallest element in a stream
#include<iostream>
#include<climits>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int heap_size; // Current number of elements in min heap
public:
    MinHeap(int a[], int size); // Constructor
    void buildHeap();
    void MinHeapify(int i);  //To minheapify subtree rooted with index i
    int parent(int i)  { return (i-1)/2;  }
    int left(int i)    { return (2*i + 1);  }
    int right(int i)   { return (2*i + 2);  }
    int extractMin();  // extracts root (minimum) element
    int getMin()       {  return harr[0]; }

    // to replace root with new node x and heapify() new root
    void replaceMin(int x) { harr[0] = x; MinHeapify(0); }
};

MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
}

void MinHeap::buildHeap()
{
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
```

```
}

// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{
    if (heap_size == 0)
        return INT_MAX;

    // Store the minimum vakue.
    int root = harr[0];

    // If there are more than 1 items, move the last item to root
    // and call heapify.
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        MinHeapify(0);
    }
    heap_size--;

    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// Function to return k'th largest element from input stream
void kthLargest(int k)
{
    // count is total no. of elements in stream seen so far
    int count = 0, x;  // x is for new element

    // Create a min heap of size k
    int *arr = new int[k];
    MinHeap mh(arr, k);


    while (1)
    {
        // Take next element from stream
        cout << "Enter next element of stream ";
        cin >> x;

        // Nothing much to do for first k-1 elements
        if (count < k-1)
        {
            arr[count] = x;
```

```
            count++;
        }

        else
        {
          // If this is k'th element, then store it
          // and build the heap created above
          if (count == k-1)
          {
              arr[count] = x;
              mh.buildHeap();
          }

          else
          {
               // If next element is greater than
               // k'th largest, then replace the root
               if (x > mh.getMin())
                  mh.replaceMin(x); // replaceMin calls
                                    // heapify()
          }

          // Root of heap is k'th largest element
          cout << "K'th largest element is "
               << mh.getMin() << endl;
          count++;
        }
    }
}

// Driver program to test above methods
int main()
{
    int k = 3;
    cout << "K is " << k << endl;
    kthLargest(k);
    return 0;
}
```

Run on IDE

Output

```
K is 3
Enter next element of stream 23
Enter next element of stream 10
Enter next element of stream 15
K'th largest element is 10
Enter next element of stream 70
K'th largest element is 15
Enter next element of stream 5
K'th largest element is 15
Enter next element of stream 80
K'th largest element is 23
Enter next element of stream 100
K'th largest element is 70
Enter next element of stream
CTRL + C pressed
```

Asked in: Cisco

Largest triplet product in a stream

Given a stream of integers represented as arr[]. For each index i from 0 to n-1, print the multiplication of largest, second largest, third largest element of the subarray arr[0…i]. If i < 2 print -1.

Examples:

```
Input : arr[] = {1, 2, 3, 4, 5}
Output :-1
        -1
         6
         24
         60
Explanation : for i = 2 only three elements
are there {1, 2, 3} so answer is 6. For i = 3
largest three elements are {2, 3, 4} their
product is 2*3*4 = 24 ....so on
```

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

We will use priority queue here.

1. Insert arr[i] in the priority queue
2. As the top element in priority queue is largest so pop it and store it as x. Now the top element in the priority queue will be the second largest element in subarray arr[0…i] pop it and store as y. Now the top element is third largest element in subarray arr[0…i] so pop it and store it as z.
3. Print x*y*z
4. Reinsert x, y, z.

```
// C++ implementation of largest triplet
// multiplication
```

```
#include<bits/stdc++.h>
using namespace std;

// Prints the product of three largest numbers
// in subarray arr[0..i]
void LargestTripletMultiplication(int arr[], int n)
{
    // call a priority queue
    priority_queue <int> q;

    // traversing the array
    for (int i = 0; i<n; i++)
    {
        // pushing arr[i] in array
        q.push(arr[i]);

        // if less than three elements are present
        // in array print -1
        if (q.size() < 3)
            cout << "-1" << endl;
        else
        {
            // pop three largest elements
            int x = q.top();
            q.pop();
            int y = q.top();
            q.pop();
            int z = q.top();
            q.pop();

            // Reinsert x, y, z in priority_queue
            int ans = x*y*z;
            cout << ans << endl;
            q.push(x);
            q.push(y);
            q.push(z);
        }
    }
    return ;
}

// Driver Function
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    LargestTripletMultiplication(arr, n);
    return 0;
}
```

Run on IDE

Output:

```
-1
-1
6
24
60
```

This article is contributed by **Ayush Jha**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to

Median in a stream of integers (running integers)

Given that integers are read from a data stream. Find median of elements read so for in efficient way. For simplicity assume there are no duplicates. For example, let us consider the stream 5, 15, 1, 3 …

```
After reading 1st element of stream - 5 -> median - 5
After reading 2nd element of stream - 5, 15 -> median - 10
After reading 3rd element of stream - 5, 15, 1 -> median - 5
After reading 4th element of stream - 5, 15, 1, 3 -> median - 4, so on...
```

Making it clear, when the input size is odd, we take the middle element of sorted data. If the input size is even, we pick average of middle two elements in sorted stream.

Note that output is *effective median* of integers read from the stream so far. Such an algorithm is called online algorithm. Any algorithm that can guarantee output of *i*-elements after processing *i*-th element, is said to be ***online algorithm***. Let us discuss three solutions for the above problem.

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

Method 1: Insertion Sort

If we can sort the data as it appears, we can easily locate median element. *Insertion Sort* is one such online algorithm that sorts the data appeared so far. At any instance of sorting, say after sorting *i*-th element, the first *i* elements of array are sorted. The insertion sort doesn't depend on future data to sort data input till that point. In other words, insertion sort considers data sorted so far while inserting next element. This is the key part of insertion sort that makes it an online algorithm.

However, insertion sort takes $O(n^2)$ time to sort *n* elements. Perhaps we can use *binary search* on *insertion sort* to find location of next element in O(log n) time. Yet, we can't do data movement in O(log n) time. No

matter how efficient the implementation is, it takes polynomial time in case of insertion sort.

Interested reader can try implementation of Method 1.

Method 2: Augmented self balanced binary search tree (AVL, RB, etc…)

At every node of BST, maintain number of elements in the subtree rooted at that node. We can use a node as root of simple binary tree, whose left child is self balancing BST with elements less than root and right child is self balancing BST with elements greater than root. The root element always holds *effective median*.

If left and right subtrees contain same number of elements, root node holds average of left and right subtree root data. Otherwise, root contains same data as the root of subtree which is having more elements. After processing an incoming element, the left and right subtrees (BST) are differed utmost by 1.

Self balancing BST is costly in managing balancing factor of BST. However, they provide sorted data which we don't need. We need median only. The next method make use of Heaps to trace median.

Method 3: Heaps

Similar to balancing BST in Method 2 above, we can use a max heap on left side to represent elements that are less than *effective median*, and a min heap on right side to represent elements that are greater than *effective median*.

After processing an incoming element, the number of elements in heaps differ utmost by 1 element. When both heaps contain same number of elements, we pick average of heaps root data as *effective median*. When the heaps are not balanced, we select *effective median* from the root of heap containing more elements.

Given below is implementation of above method. For algorithm to build these heaps, please read the highlighted code.

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

// Heap capacity
#define MAX_HEAP_SIZE (128)
#define ARRAY_SIZE(a) sizeof(a)/sizeof(a[0])

//// Utility functions

// exchange a and b
inline
void Exch(int &a, int &b)
{
    int aux = a;
    a = b;
    b = aux;
}

// Greater and Smaller are used as comparators
bool Greater(int a, int b)
{
    return a > b;
}
```

```
bool Smaller(int a, int b)
{
    return a < b;
}

int Average(int a, int b)
{
    return (a + b) / 2;
}

// Signum function
// = 0  if a == b  - heaps are balanced
// = -1 if a < b   - left contains less elements than right
// = 1  if a > b   - left contains more elements than right
int Signum(int a, int b)
{
    if( a == b )
        return 0;

    return a < b ? -1 : 1;
}

// Heap implementation
// The functionality is embedded into
// Heap abstract class to avoid code duplication
class Heap
{
public:
    // Initializes heap array and comparator required
    // in heapification
    Heap(int *b, bool (*c)(int, int)) : A(b), comp(c)
    {
        heapSize = -1;
    }

    // Frees up dynamic memory
    virtual ~Heap()
    {
        if( A )
        {
            delete[] A;
        }
    }

    // We need only these four interfaces of Heap ADT
    virtual bool Insert(int e) = 0;
    virtual int  GetTop() = 0;
    virtual int  ExtractTop() = 0;
    virtual int  GetCount() = 0;

protected:

    // We are also using location 0 of array
    int left(int i)
    {
        return 2 * i + 1;
    }

    int right(int i)
    {
        return 2 * (i + 1);
    }

    int parent(int i)
    {
        if( i <= 0 )
        {
            return -1;
        }
```

```
    return (i - 1)/2;
}

// Heap array
int   *A;
// Comparator
bool  (*comp)(int, int);
// Heap size
int   heapSize;

// Returns top element of heap data structure
int top(void)
{
    int max = -1;

    if( heapSize >= 0 )
    {
        max = A[0];
    }

    return max;
}

// Returns number of elements in heap
int count()
{
    return heapSize + 1;
}

// Heapification
// Note that, for the current median tracing problem
// we need to heapify only towards root, always
void heapify(int i)
{
    int p = parent(i);

    // comp - differentiate MaxHeap and MinHeap
    // percolates up
    if( p >= 0 && comp(A[i], A[p]) )
    {
        Exch(A[i], A[p]);
        heapify(p);
    }
}

// Deletes root of heap
int deleteTop()
{
    int del = -1;

    if( heapSize > -1)
    {
        del = A[0];

        Exch(A[0], A[heapSize]);
        heapSize--;
        heapify(parent(heapSize+1));
    }

    return del;
}

// Helper to insert key into Heap
bool insertHelper(int key)
{
    bool ret = false;

    if( heapSize < MAX_HEAP_SIZE )
    {
```

```
            ret = true;
            heapSize++;
            A[heapSize] = key;
            heapify(heapSize);
        }

        return ret;
    }
};

// Specilization of Heap to define MaxHeap
class MaxHeap : public Heap
{
private:

public:
    MaxHeap() : Heap(new int[MAX_HEAP_SIZE], &Greater)  {  }

    ~MaxHeap()  { }

    // Wrapper to return root of Max Heap
    int GetTop()
    {
        return top();
    }

    // Wrapper to delete and return root of Max Heap
    int ExtractTop()
    {
        return deleteTop();
    }

    // Wrapper to return # elements of Max Heap
    int  GetCount()
    {
        return count();
    }

    // Wrapper to insert into Max Heap
    bool Insert(int key)
    {
        return insertHelper(key);
    }
};

// Specilization of Heap to define MinHeap
class MinHeap : public Heap
{
private:

public:

    MinHeap() : Heap(new int[MAX_HEAP_SIZE], &Smaller) { }

    ~MinHeap() { }

    // Wrapper to return root of Min Heap
    int GetTop()
    {
        return top();
    }

    // Wrapper to delete and return root of Min Heap
    int ExtractTop()
    {
        return deleteTop();
    }

    // Wrapper to return # elements of Min Heap
    int  GetCount()
```

```
    {
        return count();
    }

    // Wrapper to insert into Min Heap
    bool Insert(int key)
    {
        return insertHelper(key);
    }
};

// Function implementing algorithm to find median so far.
int getMedian(int e, int &m, Heap &l, Heap &r)
{
    // Are heaps balanced? If yes, sig will be 0
    int sig = Signum(l.GetCount(), r.GetCount());
    switch(sig)
    {
    case 1: // There are more elements in left (max) heap

        if( e < m ) // current element fits in left (max) heap
        {
            // Remore top element from left heap and
            // insert into right heap
            r.Insert(l.ExtractTop());

            // current element fits in left (max) heap
            l.Insert(e);
        }
        else
        {
            // current element fits in right (min) heap
            r.Insert(e);
        }

        // Both heaps are balanced
        m = Average(l.GetTop(), r.GetTop());

        break;

    case 0: // The left and right heaps contain same number of elements

        if( e < m ) // current element fits in left (max) heap
        {
            l.Insert(e);
            m = l.GetTop();
        }
        else
        {
            // current element fits in right (min) heap
            r.Insert(e);
            m = r.GetTop();
        }

        break;

    case -1: // There are more elements in right (min) heap

        if( e < m ) // current element fits in left (max) heap
        {
            l.Insert(e);
        }
        else
        {
            // Remove top element from right heap and
            // insert into left heap
            l.Insert(r.ExtractTop());

            // current element fits in right (min) heap
            r.Insert(e);
```

```
        }

        // Both heaps are balanced
        m = Average(l.GetTop(), r.GetTop());

        break;
    }

    // No need to return, m already updated
    return m;
}

void printMedian(int A[], int size)
{
    int m = 0; // effective median
    Heap *left  = new MaxHeap();
    Heap *right = new MinHeap();

    for(int i = 0; i < size; i++)
    {
        m = getMedian(A[i], m, *left, *right);

        cout << m << endl;
    }

    // C++ more flexible, ensure no leaks
    delete left;
    delete right;
}
// Driver code
int main()
{
    int A[] = {5, 15, 1, 3, 2, 8, 7, 9, 10, 6, 11, 4};
    int size = ARRAY_SIZE(A);

    // In lieu of A, we can also use data read from a stream
    printMedian(A, size);

    return 0;
}
```

Run on IDE

Time Complexity: If we omit the way how stream was read, complexity of median finding is ***O(N log N)***, as we need to read the stream, and due to heap insertions/deletions.

At first glance the above code may look complex. If you read the code carefully, it is simple algorithm.

Median of Stream of Running Integers using STL

— **Venki**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Heap array-stream statistical-algorithms

Login to Improve this Article

Online algorithm for checking palindrome in a stream

Given a stream of characters (characters are received one by one), write a function that prints ‘Yes’ if a character makes the complete string palindrome, else prints ‘No’.

Examples:

```
Input: str[] = "abcba"
Output: a Yes   // "a" is palindrome
        b No    // "ab" is not palindrome
        c No    // "abc" is not palindrome
        b No    // "abcb" is not palindrome
        a Yes   // "abcba" is palindrome

Input: str[] = "aabaacaabaa"
Output:  a Yes   // "a" is palindrome
         a Yes   // "aa" is palindrome
         b No    // "aab" is not palindrome
         a No    // "aaba" is not palindrome
         a Yes   // "aabaa" is palindrome
         c No    // "aabaac" is not palindrome
         a No    // "aabaaca" is not palindrome
         a No    // "aabaacaa" is not palindrome
         b No    // "aabaacaab" is not palindrome
         a No    // "aabaacaaba" is not palindrome
         a Yes   // "aabaacaabaa" is palindrome
```

Let input string be str[0..n-1]. A **Simple Solution** is to do following for every character str[i] in input string. Check if substring str[0…i] is palindrome, then print yes, else print no.

A **Better Solution** is to use the idea of Rolling Hash used in Rabin Karp algorithm. The idea is to keep track of reverse of first half and second half (we also use first half and reverse of second half) for every index. Below is complete algorithm.

```
1) The first character is always a palindrome, so print yes for
   first character.
```

```
2) Initialize reverse of first half as "a" and second half as "b".
   Let the hash value of first half reverse be 'firstr' and that of
   second half be 'second'.

3) Iterate through string starting from second character, do following
    for every character str[i], i.e., i varies from 1 to n-1.
   a) If 'firstr' and 'second' are same, then character by character
      check the substring ending with current character and print
      "Yes" if palindrome.
      Note that if hash values match, then strings need not be same.
      For example, hash values of "ab" and "ba" are same, but strings
      are different. That is why we check complete string after hash.

   b) Update 'firstr' and 'second' for next iteration.
         If 'i' is even, then add next character to the beginning of
                         'firstr' and end of second half and update
                          hash values.
         If 'i' is odd,  then keep 'firstr' as it is, remove leading
                         character from second and append a next
                         character at end.
```

Let us see all steps for example string “**abcba**”

Initial values of ‘firstr’ and ‘second’

firstr’ = hash(“a”), ‘second’ = hash(“b”)

Start from second character, i.e.,

i = 1

a) Compare ‘firstr’ and ‘second’, they don’t match, so print no.

b) Calculate hash values for next iteration, i.e., i = 2

Since i is odd, ‘firstr’ is not changed and ‘second’ becomes hash(“c”)

i = 2

a) Compare ‘firstr’ and ‘second’, they don’t match, so print no.

b) Calculate hash values for next iteration, i.e., i = 3

Since i is even, ‘firstr’ becomes hash(“ba”) and ‘second’ becomes hash(“cb”)

i = 3

a) Compare ‘first’ and ‘second’, they don’t match, so print no.

b) Calculate hash values for next iteration, i.e., i = 4

Since i is odd, ‘firstr’ is not changed and ‘second’ becomes hash(“ba”)

i = 4

a) ‘firstr’ and ‘second’ match, compare the whole strings, they match, so print yes

b) We don’t need to calculate next hash values as this is last index

The idea of using rolling hashes is, next hash value can be calculated from previous in O(1) time by just doing some constant number of arithmetic operations.

Below are the implementations of above approach.

C/C++

```
// C program for online algorithm for palindrome checking
#include<stdio.h>
#include<string.h>

// d is the number of characters in input alphabet
#define d 256

// q is a prime number used for evaluating Rabin Karp's Rolling hash
#define q 103

void checkPalindromes(char str[])
{
    // Length of input string
    int N = strlen(str);

    // A single character is always a palindrome
    printf("%c Yes\n", str[0]);

    // Return if string has only one character
    if (N == 1) return;

    // Initialize first half reverse and second half for
    // as firstr and second characters
    int firstr  = str[0] % q;
    int second = str[1] % q;

    int h = 1, i, j;

    // Now check for palindromes from second character
    // onward
    for (i=1; i<N; i++)
    {
        // If the hash values of 'firstr' and 'second'
        // match, then only check individual characters
        if (firstr == second)
        {
            /* Check if str[0..i] is palindrome using
               simple character by character match */
            for (j = 0; j < i/2; j++)
            {
                if (str[j] != str[i-j])
                    break;
            }
            (j == i/2)?  printf("%c Yes\n", str[i]):
            printf("%c No\n", str[i]);
        }
        else printf("%c No\n", str[i]);

        // Calculate hash values for next iteration.
        // Don't calculate hash for next characters if
        // this is the last character of string
        if (i != N-1)
        {
            // If i is even (next i is odd)
            if (i%2 == 0)
            {
                // Add next character after first half at beginning
                // of 'firstr'
                h = (h*d) % q;
```

```
                firstr  = (firstr + h*str[i/2])%q;

                // Add next character after second half at the end
                // of second half.
                second = (second*d + str[i+1])%q;
            }
            else
            {
                // If next i is odd (next i is even) then we
                // need not to change firstr, we need to remove
                // first character of second and append a
                // character to it.
                second = (d*(second + q - str[(i+1)/2]*h)%q
                          + str[i+1])%q;
            }
        }
    }
}

/* Driver program to test above function */
int main()
{
    char *txt = "aabaacaabaa";
    checkPalindromes(txt);
    getchar();
    return 0;
}
```

Run on IDE

Java

```
// Java program for online algorithm for
// palindrome checking
public class GFG
{
    // d is the number of characters in
    // input alphabet
    static final int d = 256;

    // q is a prime number used for
    // evaluating Rabin Karp's Rolling hash
    static final int q = 103;

    static void checkPalindromes(String str)
    {
        // Length of input string
        int N = str.length();

        // A single character is always a palindrome
        System.out.println(str.charAt(0)+" Yes");

        // Return if string has only one character
        if (N == 1) return;

        // Initialize first half reverse and second
        // half for as firstr and second characters
        int firstr  = str.charAt(0) % q;
        int second = str.charAt(1) % q;

        int h = 1, i, j;

        // Now check for palindromes from second
        // character onward
        for (i = 1; i < N; i++)
        {
            // If the hash values of 'firstr' and
```

```
            // 'second' match, then only check
            // individual characters
            if (firstr == second)
            {
                /* Check if str[0..i] is palindrome
                using simple character by character
                 match */
                for (j = 0; j < i/2; j++)
                {
                    if (str.charAt(j) != str.charAt(i
                                                - j))
                        break;
                }
                System.out.println((j == i/2) ?
                  str.charAt(i) + " Yes": str.charAt(i)+
                  " No");
            }
            else System.out.println(str.charAt(i)+ " No");

            // Calculate hash values for next iteration.
            // Don't calculate hash for next characters
            // if this is the last character of string
            if (i != N - 1)
            {
                // If i is even (next i is odd)
                if (i % 2 == 0)
                {
                    // Add next character after first
                    // half at beginning of 'firstr'
                    h = (h * d) % q;
                    firstr  = (firstr + h *str.charAt(i /
                                                2)) % q;

                    // Add next character after second
                    // half at the end of second half.
                    second = (second * d + str.charAt(i +
                                                1)) % q;
                }
                else
                {
                    // If next i is odd (next i is even)
                    // then we need not to change firstr,
                    // we need to remove first character
                    // of second and append a character
                    // to it.
                    second = (d * (second + q - str.charAt(
                             (i + 1) / 2) * h) % q +
                               str.charAt(i + 1)) % q;
                }
            }
        }
    }

    /* Driver program to test above function */
    public static void main(String args[])
    {
        String txt = "aabaacaabaa";
        checkPalindromes(txt);
    }
}
// This code is contributed by Sumit Ghosh
```

Run on IDE

Python

```
# Python program Online algorithm for checking palindrome
```

Queue based approach for first non-repeating character in a stream

Given a stream of characters and we have to find first non repeating character each time a character is inserted to the stream.

Examples:

```
Input  : a a b c
Output : a -1 b b

Input  : a a c
Output : a -1 c
```

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

We have already discussed a Doubly linked list based approach in the previous post.

Approach-

1. Create a count array of size 26(assuming only lower case characters are present) and initialize it with zero.
2. Create a queue of char datatype.
3. Store each character in queue and increase its frequency in the hash array.
4. For every character of stream, we check front of the queue.
5. If the frequency of character at the front of queue is one, then that will be the first non repeating character.
6. Else if frequency is more than 1, then we pop that element.
7. If queue became empty that means there are no non repeating character so we will print -1.

C++

```
// C++ program for a Queue based approach to find first non-repeating
// character
#include <bits/stdc++.h>
using namespace std;
const int CHAR_MAX = 26;

// function to find first non repeating
// character of sa stream
void firstnonrepeating(char str[])
{
    queue<char> q;
    int charCount[CHAR_MAX] = { 0 };

    // traverse whole stream
    for (int i = 0; str[i]; i++) {

        // push each character in queue
        q.push(str[i]);

        // increment the frequency count
        charCount[str[i]-'a']++;

        // check for the non pepeating character
        while (!q.empty())
        {
            if (charCount[q.front()-'a'] > 1)
                q.pop();
            else
            {
                cout << q.front() << " ";
                break;
            }
        }

        if (q.empty())
            cout << -1 << " ";
    }
    cout << endl;
}

// Driver function
int main()
{
    char str[] = "aabc";
    firstnonrepeating(str);
    return 0;
}
```

Run on IDE

Java

```
//Java Program for a Queue based approach to find first non-repeating
//character

import java.util.LinkedList;
import java.util.Queue;

public class NonReapatingCQueue
{
     final static int CHAR_MAX = 26;
```

```
    // function to find first non repeating
    // character of stream
     static void firstNonRepeating(String str)
     {
         //count array of size 26(assuming only lower case characters are present)
         int[] charCount = new int[CHAR_MAX];

         //Queue to store Characters
         Queue<Character> q = new LinkedList<Character>();

        // traverse whole stream
         for(int i=0; i<str.length(); i++)
         {
             char c = str.charAt(i);

            // push each character in queue
             q.add(c);

            // increment the frequency count
             charCount1++;

            // check for the non repeating character
             while(!q.isEmpty())
             {
                 if(charCount[q.peek() - 'a'] > 1)
                     q.remove();
                 else
                 {
                     System.out.print(q.peek() + " ");
                     break;
                 }
             }
             if(q.isEmpty())
                 System.out.print(-1 + " ");
         }
         System.out.println();
     }

    // Driver function
    public static void main(String[] args)
    {
        String str = "aabc";
        firstNonRepeating(str);
    }

}
//This code is Contributed by Sumit Ghosh
```

Run on IDE

Output:

```
a -1 b b
```

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