Reservoir Sampling

Difficulty Level : Hard • Last Updated : 21 Oct, 2021

Reservoir sampling is a family of randomized algorithms for randomly choosing *k* samples from a list of *n* items, where *n* is either a very large or unknown number. Typically *n* is large enough that the list doesn't fit into main memory. For example, a list of search queries in Google and Facebook.
So we are given a big array (or stream) of numbers (to simplify), and we need to write an efficient function to randomly select *k* numbers where *1 <= k <= n*. Let the input array be *stream[].*

A **simple solution** is to create an array *reservoir[]* of maximum size *k*. One by one randomly select an item from *stream[0..n-1]*. If the selected item is not previously selected, then put it in *reservoir[]*. To check if an item is previously selected or not, we need to search the item in *reservoir[]*. The time complexity of this algorithm will be *O(k^2)*. This can be costly if *k* is big. Also, this is not efficient if the input is in the form of a stream.

It **can be solved in *O(n)* time**. The solution also suits well for input in the form of stream. The idea is similar to this post. Following are the steps.
1) Create an array *reservoir[0..k-1]* and copy first *k* items of *stream[]* to

current item in *stream[]*. Let the generated random number is *j*.
...**b)** If *j* is in range 0 to *k-1*, replace *reservoir[j]* with stream *[i]*

Following is the implementation of the above algorithm.

C++

```
// An efficient program to randomly select
// k items from a stream of items
#include <bits/stdc++.h>
#include <time.h>
using namespace std;

// A utility function to print an array
void printArray(int stream[], int n)
{
    for (int i = 0; i < n; i++)
        cout << stream[i] << " ";
    cout << endl;
}

// A function to randomly select
// k items from stream[0..n-1].
void selectKItems(int stream[], int n, int k)
{
    int i; // index for elements in stream[]

    // reservoir[] is the output array. Initialize
    // it with first k elements from stream[]
    int reservoir[k];
    for (i = 0; i < k; i++)
        reservoir[i] = stream[i];

    // Use a different seed value so that we don't get
    // same result each time we run this program
    srand(time(NULL));

    // Iterate from the (k+1)th element to nth element
    for (; i < n; i++)
```

```
        // If the randomly picked index is smaller than k,
        // then replace the element present at the index
        // with new element from stream
        if (j < k)
        reservoir[j] = stream[i];
    }

    cout << "Following are k randomly selected items \n";
    printArray(reservoir, k);
}

// Driver Code
int main()
{
    int stream[] = {1, 2, 3, 4, 5, 6,
                    7, 8, 9, 10, 11, 12};
    int n = sizeof(stream)/sizeof(stream[0]);
    int k = 5;
    selectKItems(stream, n, k);
    return 0;
}

// This is code is contributed by rathbhupendra
```

C

```
// An efficient program to randomly select k items from a stream of items

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// A utility function to print an array
void printArray(int stream[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", stream[i]);
    printf("\n");
}
```

```
    // reservoir[] is the output array. Initialize it with
    // first k elements from stream[]
    int reservoir[k];
    for (i = 0; i < k; i++)
        reservoir[i] = stream[i];

    // Use a different seed value so that we don't get
    // same result each time we run this program
    srand(time(NULL));

    // Iterate from the (k+1)th element to nth element
    for (; i < n; i++)
    {
        // Pick a random index from 0 to i.
        int j = rand() % (i+1);

        // If the randomly  picked index is smaller than k, then replace
        // the element present at the index with new element from stream
        if (j < k)
          reservoir[j] = stream[i];
    }

    printf("Following are k randomly selected items \n");
    printArray(reservoir, k);
}

// Driver program to test above function.
int main()
{
    int stream[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int n = sizeof(stream)/sizeof(stream[0]);
    int k = 5;
    selectKItems(stream, n, k);
    return 0;
}
```

Java

```java
public class ReservoirSampling {

    // A function to randomly select k items from
    // stream[0..n-1].
    static void selectKItems(int stream[], int n, int k)
    {
        int i; // index for elements in stream[]

        // reservoir[] is the output array. Initialize it
        // with first k elements from stream[]
        int reservoir[] = new int[k];
        for (i = 0; i < k; i++)
            reservoir[i] = stream[i];

        Random r = new Random();

        // Iterate from the (k+1)th element to nth element
        for (; i < n; i++) {
            // Pick a random index from 0 to i.
            int j = r.nextInt(i + 1);

            // If the randomly  picked index is smaller than
            // k, then replace the element present at the
            // index with new element from stream
            if (j < k)
                reservoir[j] = stream[i];
        }

        System.out.println(
            "Following are k randomly selected items");
        System.out.println(Arrays.toString(reservoir));
    }

    // Driver Program to test above method
    public static void main(String[] args)
    {
        int stream[]
            = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
        int n = stream.length;
        int k = 5;
        selectKItems(stream, n, k);
```

Python3

```
# An efficient Python3 program
# to randomly select k items
# from a stream of items
import random
# A utility function
# to print an array
def printArray(stream,n):
    for i in range(n):
        print(stream[i],end=" ");
    print();

# A function to randomly select
# k items from stream[0..n-1].
def selectKItems(stream, n, k):
        i=0;
        # index for elements
        # in stream[]

        # reservoir[] is the output
        # array. Initialize it with
        # first k elements from stream[]
        reservoir = [0]*k;
        for i in range(k):
            reservoir[i] = stream[i];

        # Iterate from the (k+1)th
        # element to nth element
        while(i < n):
            # Pick a random index
            # from 0 to i.
            j = random.randrange(i+1);

            # If the randomly picked
            # index is smaller than k,
            # then replace the element
            # present at the index
            # with new element from stream
            if(j < k):
```

```
        printArray(reservoir, k);

# Driver Code

if __name__ == "__main__":
    stream = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];
    n = len(stream);
    k = 5;
    selectKItems(stream, n, k);

# This code is contributed by mits
```

C#

```
// An efficient C# program to randomly
// select k items from a stream of items
using System;
using System.Collections;

public class ReservoirSampling
{
    // A function to randomly select k
    // items from stream[0..n-1].
    static void selectKItems(int []stream,
                             int n, int k)
    {
        // index for elements in stream[]
        int i;

        // reservoir[] is the output array.
        // Initialize it with first k
        //  elements from stream[]
        int[] reservoir = new int[k];
        for (i = 0; i < k; i++)
            reservoir[i] = stream[i];

        Random r = new Random();

        // Iterate from the (k+1)th
```

```
            int j = r.Next(i + 1);

            // If the randomly picked index
            // is smaller than k, then replace
            // the element present at the index
            // with new element from stream
            if(j < k)
                reservoir[j] = stream[i];
        }

        Console.WriteLine("Following are k " +
                    "randomly selected items");
        for (i = 0; i < k; i++)
        Console.Write(reservoir[i]+" ");
    }

    //Driver code
    static void Main()
    {
        int []stream = {1, 2, 3, 4, 5, 6, 7,
                        8, 9, 10, 11, 12};
        int n = stream.Length;
        int k = 5;
        selectKItems(stream, n, k);
    }
}

// This code is contributed by mits
```

PHP

```
<?php
// An efficient PHP program
// to randomly select k items
// from a stream of items

// A utility function
// to print an array
function printArray($stream,$n)
{
```

```php
// A function to randomly select
// k items from stream[0..n-1].
function selectKItems($stream, $n, $k)
    {
        $i; // index for elements
            // in stream[]

        // reservoir[] is the output
        // array. Initialize it with
        // first k elements from stream[]
        $reservoir = array_fill(0, $k, 0);
        for ($i = 0; $i < $k; $i++)
            $reservoir[$i] = $stream[$i];

        // Iterate from the (k+1)th
        // element to nth element
        for (; $i < $n; $i++)
        {
            // Pick a random index
            // from 0 to i.
            $j = rand(0,$i + 1);

            // If the randomly picked
            // index is smaller than k,
            // then replace the element
            // present at the index
            // with new element from stream
            if($j < $k)
                $reservoir[$j] = $stream[$i];
        }

        echo "Following are k randomly ".
                    "selected items\n";
        printArray($reservoir, $k);
    }

// Driver Code
$stream = array(1, 2, 3, 4, 5, 6, 7,
                8, 9, 10, 11, 12);
$n = count($stream);
```

```
?>
```

Javascript

```
<script>

// An efficient program to randomly select
// k items from a stream of items

// A utility function to print an array
function printArray(stream, n)
{
    for(let i = 0; i < n; i++)
        document.write(stream[i] + " ");

    document.write('\n');
}

// A function to randomly select
// k items from stream[0..n-1].
function selectKItems(stream, n, k)
{

    // Index for elements in stream[]
    let i;

    // reservoir[] is the output array. Initialize
    // it with first k elements from stream[]
    let reservoir = [];
    for(i = 0; i < k; i++)
        reservoir[i] = stream[i];

    // Use a different seed value so that
    // we don't get same result each time
    // we run this program

    // Iterate from the (k+1)th element
    // to nth element
    for(; i < n; i++)
    {
```

```
        // If the randomly picked index is
        // smaller than k, then replace the
        // element present at the index
        // with new element from stream
        if (j < k)
            reservoir[j] = stream[i];
    }

    document.write("Following are k randomly " +
                   "selected items \n");
    printArray(reservoir, k);
}

// Driver Code
let stream = [ 1, 2, 3, 4, 5, 6,
               7, 8, 9, 10, 11, 12 ];
let n = stream.length;
let k = 5;

selectKItems(stream, n, k);

// This code is contributed by rohan07

</script>
```

Output:

```
Following are k randomly selected items
6 2 11 8 12
Note: Output will differ every time as it selects and prints rando
```

Time Complexity: O(n)

Auxiliary Space: *O(k)*

How does this work?

To prove that this solution works perfectly, we must prove that the

Case 1: For last *n-k* stream items, i.e., for *stream[i]* where *k <= i < n*

For every such stream item *stream[i]*, we pick a random index from 0 to *i* and if the picked index is one of the first *k* indexes, we replace the element at picked index with *stream[i]*

To simplify the proof, let us first consider the *last item*. The probability that the last item is in final reservoir = The probability that one of the first *k* indexes is picked for last item = *k/n* (the probability of picking one of the *k* items from a list of size *n*)

Let us now consider the *second last item*. The probability that the second last item is in final *reservoir[]* = [Probability that one of the first *k* indexes is picked in iteration for *stream[n-2]*] X [Probability that the index picked in iteration for *stream[n-1]* is not same as index picked for *stream[n-2]*] = [*k/(n-1)]*[(n-1)/n*] = *k/n*.

Similarly, we can consider other items for all stream items from *stream[n-1]* to *stream[k]* and generalize the proof.

Case 2: For first *k* stream items, i.e., for *stream[i]* where *0 <= i < k*

The first *k* items are initially copied to *reservoir[]* and may be removed later in iterations for *stream[k]* to *stream[n]*.

The probability that an item from *stream[0..k-1]* is in final array = Probability that the item is not picked when items *stream[k], stream[k+1], ... stream[n-1]* are considered = *[k/(k+1)] x [(k+1)/(k+2)] x [(k+2)/(k+3)] x ... x [(n-1)/n] = k/n*

References:

http://en.wikipedia.org/wiki/Reservoir_sampling

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

Liked 30

Previous

Shuffle a given array using Fisher–Yates shuffle Algorithm

Next

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

RECOMMENDED ARTICLES

Page : 1 2

01 Check if all possible Triplet Sum is present in given Array
08, Jul 22

02 Minimize the maximum of Matrix whose rows and columns are in A.P
08, Jul 22

03 Compute nCr % p | Set 4 (Chinese Remainder theorem with Lucas Theorem)

05 Count of elements altering which changes the GCD of Array
07, Jul 22

06 Find original Array from given Array where each element is sum of prefix and suffix sum
07, Jul 22

07 Count of operations to make numbers equal by adding power of 2

Increment Decrement queries
07, Jul 22

bitwise XOR is X
05, Jul 22

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : Mithun Kumar, rathbhupendra, navaneethnvnthr, rohan07, subhammahato348

Article Tags : Articles, Mathematical, Randomized

Practice Tags : Mathematical

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS