Reservoir Sampling

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.

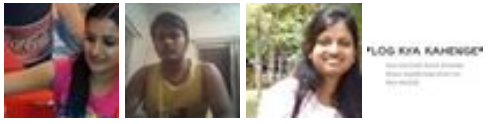
It **can be solved in *O(n)* time**. The solution also suits well for input in the form of stream. The idea is similar to this post. Following are the steps.

1) Create an array *reservoir[0..k-1]* and copy first *k* items of *stream[]* to it.
2) Now one by one consider all items from *(k+1)*th item to *n*th item.
…**a)** Generate a random number from 0 to *i* where *i* is index of 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 *arr[i]*

Following is C implementation of the above algorithm.

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

#include <stdio.h>
```

```c
#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");
}

// 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++)
    {
        // Pick a random index from 0 to i.
        int j = rand() % (i+1);

        // If the randomly  picked index is smaller than k, then repla
        // the element present at the index with new element from stre
        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]);
```


Popular Posts

```
    int k = 5;
    selectKItems(stream, n, k);
    return 0;
}
```

Output:

```
 Following are k randomly selected items
6 2 11 8 12
```

Time Complexity: O(n)

How does this work?
To prove that this solution works perfectly, we must prove that the probability that any item *stream[i]* where *0 <= i < n* will be in final *reservoir[]* is *k/n*. Let us divide the proof in two cases as first *k* items are treated differently.

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.

Related Tpoics:

- Analysis of Algorithms | Set 4 (Analysis of Loops)
- Analysis of Algorithms | Set 3 (Asymptotic Notations)
- NP-Completeness | Set 1 (Introduction)
- Static and Dynamic Libraries | Set 1
- The Ubiquitous Binary Search | Set 1
- Analysis of Algorithms | Set 2 (Worst, Average and Best Cases)
- Analysis of Algorithms | Set 1 (Asymptotic Analysis)
- Scope rules in C

Recent Comments

Abhi You live US or India?
Google (Mountain View) interview · 15 minutes ago

Aman Hi, Why arent we checking for conditions...
Write a C program to Delete a Tree. · 55 minutes ago

kzs please provide solution for the problem...
Backtracking | Set 2 (Rat in a Maze) · 58 minutes ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...
Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...
Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...
Find subarray with given sum · 1 hour ago

Writing code in comment? Please use **ideone.com** and share the link here.

8 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

sachin • 9 months ago

Can't we just do like this :

1.Generate random numer 0 to i (where i runs from n-1 to n-k)
2. Replace ith element with random generated index.
At the end, last k elements will have random permutation.
It is O(k) time complexity.

Each element has probability k/n or am I missing something?

sachin sachin • 9 months ago

This wont work for the stream though

```
/* Paste your code here (You may delete these lines if not wri
```

ANONYMOUS • a year ago

For Interested people:
http://gregable.com/2007/10/re...

```
/* Paste your code here (You may delete these lines if not writing c
```

1 ^ | v •

rahul • 2 years ago

in this part :-
[Probability that one of the first k indexes is picked in iteration for stream[n-2]] X
iteration for stream[n-1] is not same as index picked for stream[n-2]] = [k/(n-1

[Probability that one of the first k indexes is picked in iteration for stream[n-2]] =

so,
Shouldn't it be = [k/(n-2)]*[(n-1)/n]

```
/* Paste your code here (You may delete these lines if not writing c
```

^ | v •

sandy • 2 years ago

I think the output can never have 4 in first place. Can you tell me how it can ha

```
/* Paste your code here (You may delete these lines if not writing c
```

^ | v •

GeeksforGeeks ➚ sandy • 2 years ago

Well pointed out :) We manually wrote the output as it was just 5 randc
We have now placed output of a run.

^ | v •

kiran • 2 years ago

I think srand() function can be outside of the loop.

```
/* Paste your code here (You may delete these lines if not writing c
```

GeeksforGeeks ➚ kiran • 2 years ago

Thanks for suggesting the optimization. We have updated the post.

Subscribe Add Disqus to your site