Previous Next

470. Implement Rand10() Using Rand7()

July 18, 2018 | 24K views

Average Rating: 4.22 (23 votes)

Given a function `rand7` which generates a uniform random integer in the range 1 to 7, write a function `rand10` which generates a uniform random integer in the range 1 to 10.

Do NOT use system's `Math.random()`.

Example 1:

```
Input: 1
Output: [7]
```

Example 2:

```
Input: 2
Output: [8,4]
```

Example 3:

```
Input: 3
Output: [8,1,10]
```

Note:

1. `rand7` is predefined.
2. Each testcase has one argument: `n`, the number of times that `rand10` is called.

Follow up:

1. What is the expected value for the number of calls to `rand7()` function?
2. Could you minimize the number of calls to `rand7()`?

Solution

Approach 1: Rejection Sampling

Intuition

What if you could generate a random integer in the range 1 to 49? How would you generate a random integer in the range of 1 to 10? What would you do if the generated number is in the desired range? What if it is not?

Algorithm

This solution is based upon Rejection Sampling. The main idea is when you generate a number in the desired range, output that number immediately. If the number is out of the desired range, reject it and re-sample again. As each number in the desired range has the same probability of being chosen, a uniform distribution is produced.

Obviously, we have to run rand7() function at least twice, as there are not enough numbers in the range of 1 to 10. By running rand7() twice, we can get integers from 1 to 49 uniformly. Why?

	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	8	9	10	1	2	3	4
3	5	6	7	8	9	10	1
4	2	3	4	5	6	7	8
5	9	10	1	2	3	4	5
6	6	7	8	9	10	*	*
7	*	*	*	*	*	*	*

A table is used to illustrate the concept of rejection sampling. Calling rand7() twice will get us row and column index that corresponds to a unique position in the table above. Imagine that you are choosing a number randomly from the table above. If you hit a number, you return that number immediately. If you hit a *, you repeat the process again until you hit a number.

Since 49 is not a multiple of 10, we have to use rejection sampling. Our desired range is integers from 1 to 40, which we can return the answer immediately. If not (the integer falls between 41 to 49), we reject it and repeat the whole process again.

C++ Java Copy

```
class Solution {
public:
    int rand10() {
        int row, col, idx;
        do {
            row = rand7();
            col = rand7();
            idx = col + (row - 1) * 7;
        } while (idx > 40);
        return 1 + (idx - 1) % 10;
    }
};
```

Complexity Analysis

- Time Complexity: $O(1)$ average, but $O(\infty)$ worst case.

The expected value for the number of calls to rand7() can be computed as follows:

$$
\begin{aligned}
E(\#\ \text{calls to rand7}) = \ & 2 \cdot \frac{40}{49} + \\
& 4 \cdot \frac{9}{49} \cdot \frac{40}{49} + \\
& 6 \cdot \left(\frac{9}{49}\right)^2 \cdot \frac{40}{49} + \\
& \cdots \\
= \ & \sum_{k=1}^{\infty} \left(\frac{9}{49}\right)^{k-1} \cdot \frac{40}{49} \\
= \ & \frac{80}{49 \cdot \left(1 - \frac{9}{49}\right)^2} \\
= \ & 2.45
\end{aligned}
$$

- Space Complexity: $O(1)$.

Approach 2: Utilizing out-of-range samples

Intuition

There are a total of 2.45 calls to rand7() on average when using approach 1. Can we do better? Glad that you asked. In fact, we are able to improve average number of calls to rand7() by about 10%.

The idea is that we should not throw away the out-of-range samples, but instead use them to increase our chances of finding an in-range sample on the successive call to rand7.

Algorithm

Start by generating a random integer in the range 1 to 49 using the aforementioned method. In the event that we could not generate a number in the desired range (1 to 40), it is equally likely that each number of 41 to 49 would be chosen. In other words, we are able to obtain integers in the range of 1 to 9 uniformly. Now, run rand7() again to obtain integers in the range of 1 to 63 uniformly. Apply rejection sampling where the desired range is 1 to 60. If the generated number is in the desired range (1 to 60), we return the number. If it is not (61 to 63), we at least obtain integers of 1 to 3 uniformly. Run rand7() again to obtain integers in the range of 1 to 21 uniformly. The desired range is 1 to 20, and in the unlikely event we get a 21, we reject it and repeat the entire process again.

C++ Java Copy

```
class Solution {
public:
    int rand10() {
        int a, b, idx;
        while (true) {
            a = rand7();
            b = rand7();
            idx = b + (a - 1) * 7;
            if (idx <= 40)
                return 1 + (idx - 1) % 10;
            a = idx - 40;
            b = rand7();
            // get uniform dist from 1 - 63
            idx = b + (a - 1) * 7;
            if (idx <= 60)
                return 1 + (idx - 1) % 10;
            a = idx - 60;
            b = rand7();
            // get uniform dist from 1 - 21
            idx = b + (a - 1) * 7;
            if (idx <= 20)
                return 1 + (idx - 1) % 10;
        }
    }
};
```

Complexity Analysis

- Time Complexity: $O(1)$ average, but $O(\infty)$ worst case.

The expected value for the number of calls to rand7() can be computed as follows (with some steps omitted due to tediousness):

$$
\begin{aligned}
E(\#\ \text{calls to rand7}) = \ & 2 \cdot \frac{40}{49} + \\
& 3 \cdot \frac{9}{49} \cdot \frac{60}{63} + \\
& 4 \cdot \frac{9}{49} \cdot \frac{3}{63} \cdot \frac{20}{21} + \\
& \left(\frac{9}{49} \cdot \frac{3}{63} \cdot \frac{1}{21}\right) \times \\
& \left(6 \cdot \frac{40}{49} + 7 \cdot \frac{9}{49} \cdot \frac{60}{63} + 8 \cdot \frac{9}{49} \cdot \frac{3}{63} \cdot \frac{20}{21}\right) + \\
& \left(\frac{9}{49} \cdot \frac{3}{63} \cdot \frac{1}{21}\right)^2 \times \\
& \left(10 \cdot \frac{40}{49} + 11 \cdot \frac{9}{49} \cdot \frac{60}{63} + 12 \cdot \frac{9}{49} \cdot \frac{3}{63} \cdot \frac{20}{21}\right) + \\
& \cdots \\
= \ & 2.2123
\end{aligned}
$$

- Space Complexity: $O(1)$.

Rate this article:

Previous Next

Comments: 15

Sort By

Type comment here... (Markdown is supported)

Preview Post

vadimtukaev 119 July 21, 2018 1:21 AM

First approach can be done much more effective:

```
int
rand10(void)
```

Read More

13 Share Reply

SHOW 6 REPLIES

Rooppesh 5 May 31, 2020 10:13 AM

Why won't rand7()+rand7()/2 work?

1 Share Reply

SHOW 2 REPLIES

Kelvos 1 February 17, 2020 12:51 AM

the formatting in the solution page is a bit off if anyone can give it an update

1 Share Reply

sanazk 3 November 6, 2018 10:45 AM Report

Can someone explain to me how the table works? I can't figure out what the numbers are and how the table is filled out?

1 Share Reply

SHOW 2 REPLIES

ckclark 27 October 16, 2018 10:17 AM

The expected value of Approach 2 should be 3158401 / 1440000 = 2.193334027777778

1 Share Reply

SHOW 2 REPLIES

abdlwahdsa 0 January 13, 2019 2:25 AM

a simpler way to this once the first 2 rand7() calls give a number in [41,49] is to do 1 more rand7() and think of the 3 calls so far as a 3-dim index into a 7x7x7 cube. if it falls in [1,340] then we're good. this way we end up with the same passing probability of 340/(7x7x7) = 340/343.
This can be generalized more easily i think. (higher dimensional cubes)

0 Share Reply

jstein 1 March 27, 2019 3:24 PM

Just notice 7 * 7 * 7 * 7 = 2401, and we know the prob of having to call rand10() again is just 1/2401. Though the worst time complexity is still O(inf), the prob of worse cases are much smaller (compared to the prob in Approach 1, which is 9/49).

0 Share Reply

SHOW 1 REPLY

maliksagar 11 February 25, 2019 11:56 AM

Simple , concise and can be extended for any k

```
class Solution extends SolBase {
   static final int K=10;
  int curr=rand7();
```

Read More

0 Share Reply

SHOW 4 REPLIES

aniket978 11 July 18, 2018 11:14 PM Report

Complexity analysis is left blank. Is it a mistake?

0 Share Reply

SHOW 1 REPLY

awaracoder 13 June 3, 2020 10:16 AM

Copy the expected value explanation to a MathJax editor to render it.

0 Share Reply

‹ 1 2 ›