Make a fair coin from a biased coin

You are given a function foo() that represents a biased coin. When foo() is called, it returns 0 with 60% probability, and 1 with 40% probability. Write a new function that returns 0 and 1 with 50% probability each. Your function should use only foo(), no other library method.

Solution:
We know foo() returns 0 with 60% probability. How can we ensure that 0 and 1 are returned with 50% probability?
The solution is similar to this post. If we can somehow get two cases with equal probability, then we are done. We call foo() two times. Both calls will return 0 with 60% probability. So the two pairs (0, 1) and (1, 0) will be generated with equal probability from two calls of foo(). Let us see how.

(0, 1): The probability to get 0 followed by 1 from two calls of foo() = 0.6 * 0.4 = 0.24
(1, 0): The probability to get 1 followed by 0 from two calls of foo() = 0.4 * 0.6 = 0.24

So the two cases appear with equal probability. The idea is to return consider only the above two cases, return 0 in one case, return 1 in other case. For other cases [(0, 0) and (1, 1)], recur until you end up in any of the above two cases.

The below program depicts how we can use foo() to return 0 and 1 with equal probability.

```
#include <stdio.h>

int foo() // given method that returns 0 with 60% probability and 1 wi
{
    // some code here
}

// returns both 0 and 1 with 50% probability
```

Google™ Custom Search

GeeksforGeeks

53,525 people like GeeksforGeeks.

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

```
int my_fun()
{
    int val1 = foo();
    int val2 = foo();
    if (val1 == 0 && val2 == 1)
        return 0;   // Will reach here with 0.24 probability
    if (val1 == 1 && val2 == 0)
        return 1;   // // Will reach here with 0.24 probability
    return my_fun();  // will reach here with (1 - 0.24 - 0.24) probab
}

int main()
{
    printf ("%d ", my_fun());
    return 0;
}
```

References:
http://en.wikipedia.org/wiki/Fair_coin#Fair_results_from_a_biased_coin

This article is compiled by **Shashank Sinha** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.
If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C

JDBC to Informix

progress.com/Informix

Supports Latest Data Connections J2EE Certified. Download Eval Now!

- Understanding "extern" keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Related Tpoics:

- Backtracking | Set 8 (Solving Cryptarithmetic Puzzles)
- Tail Recursion
- Find if two rectangles overlap
- Analysis of Algorithm | Set 4 (Solving Recurrences)
- Print all possible paths from top left to bottom right of a mXn matrix
- Generate all unique partitions of an integer
- Russian Peasant Multiplication
- Closest Pair of Points | O(nlogn) Implementation

2 Tweet 0 0

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

9 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

khatamNAAYAK • a year ago

hey, See the diff in prob in both cases i.e. getting 0 or 1 from foo() is 0.2. So, b
perfectness. That means, the solutions would be perfect if they are flipped in j
Here's my solution:

1. Take a no. from foo say x
2. Generate a random no rand from 0 to 9
3. If rand is equal to 0:

5. return x

X is now a perfect output with prob 1 = 0.5 and prob 0 = 0.5.

Please do comment back if I missed something....

avinash · 2 years ago

while loop could be better instead of recursion because it might be case of sta

1

Divesh · 2 years ago

int myfoo()
if(_foo()==0)
return 1;
if(_foo()==1)
return 0;

int requiredFoo()

if(myfoo() ^ foo() == 1) // probability 2*.24
return 1;
elseif(myfoo() ^ foo() ==0) // probability 2 * .24
return 0;
else
return requiredFoo(); // probability .04

ramesh · 2 years ago

Similar solution, but I solved it using XOR

705

Subscribe

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

```
/* Paste your code here (You may delete these lines if not writing co
foo(){

}

int bar(){
    int a = foo();
    int b = foo();
    c = a^b;
    if(c){
        return a;  // a is 0 or 1 with equal probability
    } else {
        return bar();
    }
}
```

Aashish • 2 years ago

Perhaps we can reduce the conditional checks with a little modification.

```
int my_fun()
{
    int val1 = foo();
    int val2 = 1 - foo();

    if( val1 == val2 ) // probability p * ( 1 - p )
        return val1;

    return my_fun();
}
```

aleks_misyuk • 2 years ago

Maybe it's more effectively.
This solution requires one foo() call.

```
// returns both 0 and 1 with 50% probability
int my_fun() {
    int val1 = foo();
    int val2 = foo();
    while (val1 == val2) {
        val1 = val2;
        val2 = foo();
    return val1;
}
```

Hongliang • 2 years ago

em, this one does not work, actually. For example, we made two tosses, they fails. One can start over, but this algorithm can not guarantee the result.

Some good solutions can be found online. The famous one is from John von N
http://en.wikipedia.org/wiki/F...

"

Toss the coin twice.
If the results match, start over, forgetting both results.
If the results differ, use the first result, forgetting the second.
"

http://stackoverflow.com/quest...

It is not so efficient. There are some further discussions, for example, this tree (almost) solution:

http://web.eecs.umich.edu/~qst...

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

Raguu ➦ Hongliang • 2 years ago

The method suggested on wiki page is same as the solution by @Sha

Hari • 2 years ago

Awesome

Subscribe   Add Disqus to your site