FiveThirtyEight's February 18, 2022 Riddler

Emma Knight

February 20, 2022

This week's riddler, courtesy of Gary Yane, is about forcing a coin to become lucky:

Question 1. *I have in my possession 1 million fair coins. Before you ask, these are not legal tender. Among these, I want to find the "luckiest" coin.*

I first flip all 1 million coins simultaneously (I'm great at multitasking like that), discarding any coins that come up tails. I flip all the coins that come up heads a second time, and I again discard any of these coins that come up tails. I repeat this process, over and over again. If at any point I am left with one coin, I declare that to be the "luckiest" coin.

But getting to one coin is no sure thing. For example, I might find myself with two coins, flip both of them and have both come up tails. Then I would have zero coins, never having had exactly one coin.

What is the probability that I will - at some point - have exactly one "luckiest" coin?

Let $p(n)$ be the probability that Gary has a lucky coin, given that he starts with n coins. We want to compute $p(1000000)$.

One sees from the definition that $p(0) = 0$ and $p(1) = 1$[1]. For $n \geq 2$, one has that $p(n) = \sum_{k=0}^{n} \binom{n}{k} \frac{f(k)}{2^k}$, or that $p(n) = \sum_{k=0}^{n-1} \binom{n}{k} \frac{f(k)}{2^k - 1}$. This formula lets one recursively compute $p(n)$, and doing so in python shows that the values seem to stablize at a number that is roughly .72135.

I have no further insight into this; I tried to solve this recursion and got absolutely nowhere. I'll conclude with my code:

```
import math

##This computes the values of p(n) up to top, with
##values being the, well, values.
```

[1] Sadly, the obvious continuation of this doesn't work

```
values = [0, 1]
top = 1000

for n in range(2, top):
    s = 0
    for k in range(n):
        s += values[k]*math.comb(n, k)/(2**n-1)
    values.append(s)

print(values)
```