FiveThirtyEight's June 24, 2022 Riddler

Emma Knight

June 25, 2022

This week's riddler, courtesy of Quoc Tran, is about goats in a building:

Question 1. *A goat tower has 10 floors, each of which can accommodate a single goat. Ten goats approach the tower, and each goat has its own (random) preference of floor. Multiple goats can prefer the same floor.*

One by one, each goat walks up the tower to its preferred room. If the floor is empty, the goat will make itself at home. But if the floor is already occupied by another goat, then it will keep going up until it finds the next empty floor, which it will occupy. But if it does not find any empty floors, the goat will be stuck on the roof of the tower.

What is the probability that all 10 goats will have their own floor, meaning no goat is left stranded on the roof of the tower?

To build some intuition, I will think about various smaller versions of the problem. The one goat problem is trivial (the goat always gets to their floor). With two goats, they are fine unless both prefer the second floor, which means that there is a three in four chance that no goat gets stuck on the roof of the floor. The three goat problem is harder, so I will write out the probability of success based on which floor goats are on; I will encode this with a number in binary, so that 100 means that there is a goat on the third floor and no goats on the second or first floors.

Goats	Probability
111	1
110	1
101	$\frac{2}{3}$
100	$\frac{5}{9}$
011	$\frac{1}{3}$
010	$\frac{7}{9}$
001	$\frac{4}{9}$
000	$\frac{16}{27}$

It's not too hard to see how these numbers are computed recursively, and it's not too hard to write code to do this for arbitrary values of 10. Doing this, one gets a sequence $1, \frac{3}{4}, \frac{16}{27}, \frac{125}{256}, \frac{1296}{3025}, \ldots$. It becomes abundantly clear that the pattern is $\frac{(n+1)^{n-1}}{n^n}$. The denominator makes sense, but I have no idea why the numerator is what it is. This is easily computed to be .2357947691 for the original problem.

Here is my code in all of it's bitwhacky[1] glory:

```
##This checks to see if you can put a goat in that prefers
##room j.  If you can, it returns an array whose 0th entry
##is True and whose first entry is the code of the new tower.
##If you can't, it returns an array whose 0th term is False.
def insert(i, j, k):
    p = i
    while(p < k):
        if (2**p ^ j) == (2**p + j):
            return([True, 2**p + j])
        p += 1
    return([False])

##This is the main loop that computes the probabilities
##recursively.  Technically, since you know that the
##denominator of a tower with a open rooms is n^a, this
##actually computes the probability times n^a so that you
##don't have to deal with floats and rounding errors and
##so on.
n = 10

probs = [0]*(2**n)
probs[2**n - 1] = 1

for i in range(2**n - 2, -1, -1):
    tot = 0
    for j in range(n):
        a = insert(j, i, n)
        if a[0]:
            tot += probs[a[1]]
    probs[i] = tot

print(probs)
```

[1]While I don't like bitwhacking, this is a stunningly obvious way to make the symbol table that I need, and it is supremely fast so I don't really see a nice alternative.