UMMC

Question 8

Jiamu Li & Frank Tang & Edward Wang

July 24, 2022

Contents

A particle generator is emitting two types of particles (called X and Y) into a long tube. The particles will line up in order after entering the tube. Initially, the tube is empty. At each shot, either an X- or Y-particle is randomly emitted into the tube with equal probability. Different shots are assumed to be independent from each other. Suppose that n shots have been emitted.

PROBLEM 1

What is the probability that no two X-particles are next to each other?

THEOREM 1. *The probability that no two X-particles are next to each other after n shots is given by*

$$\frac{F_{n+2}}{2^n},$$

where F_n is the nth Fibonacci number.

Proof. The probability we require can be calculated by dividing the total number of ways to arrange the contents of the tube such that there are no consecutive X-particles, by the total number of arrangements of the tube. That is to say:

$$\Pr(\text{No consecutive X-particles}) = \frac{\#\text{Arrangements w/o consecutive X-particles}}{\#\text{Total arrangements}}$$

CLAIM 1.1. *The number of arrangements with no consecutive X-particles is*

$$\sum_{k=0}^{n} \binom{n-k+1}{k}.$$

Proof. Consider a tube with n particles in it. Let the number of X-particles be equal to k, and the number of Y-particles be equal to $n-k$. Consider the tube without the X-particles, consisting solely of Y-particles in a line:

$$\underbrace{\text{YY}\ldots\text{YY}}_{n-k} \tag{1}$$

Now consider the 'gaps' between these Y-particles, indicated by a bar (|):

$$|\text{Y}|\text{Y}|\ldots|\text{Y}|\text{Y}| \tag{2}$$

Notice that there are exactly $n-k+1$ 'gaps'. Clearly, if we were to only place X-particles in the gaps, then there would never be any consecutive X-particles. This can be done in a total of

$$\binom{n-k+1}{k}$$

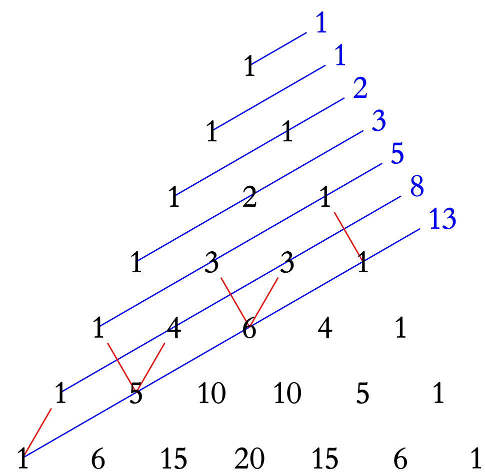

Figure 1

ways. However, we must consider this for any number of X-particles k, so we arrive at the sum

$$\#\text{Arrangements with no consecutive X-particles} = \sum_{k=0}^{n} \binom{n-k+1}{k}. \qquad \square$$

Claim 1.2. *We claim that*

$$\sum_{k=0}^{n} \binom{n-k+1}{k} = F_{n+2},$$

where F_n is the nth Fibonnaci number.

Proof. Figure 1 showcases a way to obtain the identity from Pascal's triangle. We will present an algebraic proof below.

Recall that the Fibonnaci numbers are defined as follows:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \qquad n > 1 \end{aligned}$$

Let $f(x) := \sum_{k=0}^{x} \binom{x-k+1}{k}$. It is sufficient to prove that $f(1) = F_3 = 2$, $f(2) = F_4 = 3$, and that $f(n) = f(n-1) + f(n-2)$, which would then imply the result by definition of the Fibonnaci numbers.

It is obvious that $f(1) = \binom{2}{0} + \binom{1}{1} = 2$, which is equal to F_3. Next, $f(2) = \binom{3}{0} + \binom{2}{1} + \binom{1}{2} = 3$. Notice that we define $\binom{n}{k} = 0$ when $n < k$, as it is impossible to choose k things from a set with elements less than k.

We proceed to prove that $f(n) = f(n-1) + f(n-2)$, where $n > 2$.

Using the fact that $\binom{n}{0} = 1$, we rewrite $f(n)$ using Pascal's identity and linearity as

$$f(n) = 1 + \sum_{k=1}^{n} \binom{n-k+1}{k} = 1 + \sum_{k=1}^{n} \binom{n-k}{k} + \sum_{k=1}^{n} \binom{n-k}{k-1}.$$

Next, we simplify, getting

$$\begin{aligned}
f(n) &= \sum_{k=0}^{n} \binom{n-k}{k} + \sum_{k=1}^{n} \binom{n-k}{k-1} \\
&= \sum_{k=0}^{n-1} \binom{n-k}{k} + \binom{n-n}{k} + \sum_{k=1}^{n} \binom{n-k}{k-1} \\
&= \sum_{k=0}^{n-1} \binom{n-k}{k} + 0 + \sum_{k=0}^{n-1} \binom{n-k-1}{k} \\
&= \sum_{k=0}^{n-1} \binom{n-k}{k} + \sum_{k=0}^{n-2} \binom{n-k-1}{k} + \binom{n-(n-1)-1}{n-1} \\
&= f(n-1) + f(n-2) + 0.
\end{aligned}$$

Hence $f(n)$ must be equivalent to F_{n+2}. □

Claim 1.3. *The number of total arrangements of a tube with n particles is*

$$2^n.$$

Proof. Each particle in the tube can be either an X-particle or a Y-particle, meaning there are 2 choices for each of the n particles. Hence, there are a total of 2^n arrangements. □

Hence by dividing the number of arrangements where there are no consecutive X-particles by the total number of arrangements, we arrive at the formula

$$\frac{F_{n+2}}{2^n}$$

which gives the desired probability. □

PROBLEM 2

Compute the average number of particles for $n = 2, 3, 4$.

Can you find the pattern and establish an explicit formula for general n?

THEOREM 2. *The average number of particles after n shots is*

$$0.75n + 0.25.$$

Proof. Let the average number of particles after n shots be T_n. Obviously, $T_1 = 1$. Next, consider the chance that after firing a shot, the number of particles *doesn't* decrease. This occurs only in the event that the last particle in the tube is an X-particle, and when the particle emitted is also an X-particle. Since both events have a probability 0.5, the probability that both occur is simply 0.25. Hence, the probability that the number of particles *does* increase after firing a shot is $1 - 0.25 = 0.75$. Thus the expected number of particles increases by 0.75 after each shot, giving us the recursion

$$T_{n+1} = T_n + 0.75.$$

Since $T_1 = 1$, we arrive at the formula

$$T_n = 0.75n + 0.25.$$ □

PROBLEM 3

Suppose that at each shot, an X-particle is emitted with probability $p \in (0, 1)$.

Under the same assumption as above, when n is very large, do you think the proportion of X-particles in the tube will eventually stabilise at a certain number? Why/why not?

If so, can you compute this number explicitly?

THEOREM 3. *The limiting ratio of X-particles to total particles in the tube, where the probability of emitting an X-particle is $p \in (0, 1)$, as the number of shots approaches infinity is*

$$\frac{p}{1+p}.$$

Proof. We use similar logic as done in Theorem 2 to arrive at a formula for the expected number of particles after n shots when the probability is p.

CLAIM 3.1. *The average number of particles after n shots when the probability of emitting an X-particle is $p \in (0, 1)$ is*

$$(1 - p^2)n + p^2.$$

Proof. Again, let the average number of particles after n shots be T_n. The number of particles *doesn't* increase when firing an X-particle and the last particle is also an X-particle, which has a probabilty of p^2 of happening. Hence, the probability that the number of parrticles *does* increase is $1 - p^2$, meaning that the expected number of particles increases by $1 - p^2$ after each shot, and thus we obtain

$$T_{n+1} = T_n + (1 - p^2).$$

Since $T_1 = 1$, the formula for T_n is

$$T_n = (1 - p^2)n + p^2. \qquad \square$$

We can now find the expected number of X-particles in the tube. Since the expected number of Y-particles in the tube is simply $(1 - p)n$, because the probability of emitting a Y-particle is $(1 - p)$ at each shot, the expected number of X-particles is

$$\begin{aligned}
\#\text{X-particles} &= \#\text{Total particles} - \#\text{Y-particles} \\
&= T_n - (1 - p)n \\
&= (1 - p^2)n + p^2 - (1 - p)n.
\end{aligned}$$

We then divide this by the total number of particles to obtain the desired proportion and then take the limit as $n \to \infty$, getting

$$\lim_{n\to\infty} \frac{(1 - p^2)n + p^2 - (1 - p)n}{(1 - p^2)n + p^2} = \lim_{n\to\infty} \frac{(1 - p^2) + \frac{p^2}{n} - (1 - p)}{(1 - p^2) + \frac{p^2}{n}}$$

By the algebraic limit theorem, the terms with a denominator of n become neglibibly small, and so we obtain

$$\begin{aligned}\lim_{n\to\infty}\frac{(1-p^2)+0-(1-p)}{(1-p^2)+0} &= \lim_{n\to\infty}\frac{p-p^2}{1-p^2}\\ &= \lim_{n\to\infty}\frac{p(1-p)}{(1+p)(1-p)}\\ &= \lim_{n\to\infty}\frac{p}{1-p}\end{aligned}$$

Since the limit no longer has any terms containing n, we can remove the limit, getting the final result

$$\frac{p}{1-p}.$$

□

Code Implementation

```
#include <chrono>
#include <future>
#include <iostream>
#include <random>
#include <string>
#include <thread>
#include <vector>

using namespace std::chrono;
using namespace std;

random_device rd;
mt19937 rng(rd());

const char particles[] = "XY";

unsigned int num_threads = thread::hardware_concurrency();

string gen_tube(int length) {
  uniform_int_distribution<int> pick(0, 1);
  string tube;
  for (int i = 0; i < length; i++) {
    tube += particles[pick(rng)];
  }
  return tube;
}

string annihilate(string tube) {
  string output = "";
  for (int i = 0; i < tube.length() - 1; i++) {
    if (tube[i] != tube[i + 1] && tube[i] == 'X')
      output += tube[i];
    else if (tube[i] != 'X')
      output += tube[i];
  }
  output.push_back(tube.back());
  return output;
}

bool check_consec_x(string tube) {
  for (int i = 0; i < tube.length() - 1; i++) {
    if (tube[i] == tube[i + 1])
      return 0;
  }
  return 1;
}

double prob_no_consec_after_n(int n, int runs) { // Q1
  int count = 0;
  for (int i = 0; i < runs; i++) {
    if (check_consec_x(gen_tube(n)))
      count++;
  }
  return double(count) / runs;
}
```

```cpp

double average_len_after_n(int n, int runs) { // Q2
  long long count = 0;
  for (int i = 0; i < runs; i++) {
    count += annihilate(gen_tube(n)).length();
  }
  return count / double(runs);
}

double threaded_avg_len_after_n(int n, int runs) {
  double avg = 0;
  vector<future<double>> threads;
  for (int i = 0; i < num_threads; i++) {
    threads.push_back(
        async(launch::async, average_len_after_n, n, runs / num_threads));
  }
  for (auto &t : threads) {
    avg += t.get();
  }
  return avg / num_threads;
}

double proportion_x(int n) {
  string tube = "";
  vector<future<string>> threads;
  for (int i = 0; i < num_threads; i++) {
    string small_tube = gen_tube(n / num_threads);
    threads.push_back(async(launch::async, annihilate, small_tube));
  }
  for (auto &t : threads) {
    tube += t.get();
  }
  int len = tube.length();
  return (len - 0.5 * n) / len;
}

int main() {
  auto start = high_resolution_clock::now();

  vector<future<double>> threads;
  unsigned long long runs = 10000000;

  // double avg = 0;
  // for (int i = 0; i < num_threads; i++) {
  //   threads.push_back(async(launch::async, average_len_after_n, 10,
  //   runs/num_threads));
  // }
  // for (auto &t : threads) {
  //   avg += t.get();
  // }
  // for (int i = 1; i <= 11; i++) {
  //   cout << threaded_avg_len_after_n(i, runs) << '\n';
  // }

  // int count = 0;
  // string tube = annihilate(gen_tube(100000));
  // for (char &c : tube) {
```

```
  //   if (c == 'Y')
  //     count++;
  // }

  cout << proportion_x(runs) << '\n';

  auto stop = high_resolution_clock::now();
  auto elapsed = duration_cast<milliseconds>(stop - start);
  cout << elapsed.count() << "ms\n";
}
```