CSCI 1300/5001 – Fall 2019 Handout: Random Processes
Introduction to Object-Oriented Programming
Michael H. Goldwasser
Saint Louis University Monday, 7 October 2019

The Coupon Collector's Problem

We consider a probabilistic experiment known commonly as the "Coupon Collector's Problem". Its name dates back to an era in which many consumer products (e.g. a box of cereal, a pack of cigarettes) would come with a special "prize" (e.g. a coupon, a toy, a baseball card) chosen from among a collection of n distinct prize variants.

Mathematicians have studied many interesting questions that arise in such a scenario, but one of the classic such questions is how many boxes would you expect to open before you have at least one of each of the n distinct prize types. Another interesting question is what is the largest number of the *same* prize you expect to have collected by the time you first complete the set. While questions such as this can be answered by purely mathematical analyses, our goal today is to use computer simulations of the experiment to estimate these probabilities.

For the sake of this activity, we assume that the n prize types are numbered 0 through $n-1$, and that each collected prize is equally likely to be any one of those prize types. We will rely on calls to the function `randrange(n)` from Python's `random` module, as each call to that function produces a uniformly (pseudo)random selection from the range $\{0, 1, 2, \ldots, n-1\}$.

Warmup

Assume that we do a trial of this experiment with $n = 8$ (and thus prizes 0 through 7) and that the random number generator produces the following sequence of results:

$$4, 0, 4, 5, 6, 5, 7, 6, 6, 2, 3, 5, 3, 3, 3, 7, 4, 4, 3, 2, 4, 2, 4, 2, 2, 1, 4, 7, 1, 5$$

1. How many selections were required before a complete set of values had been collected?

2. At the time the collection was first completed, which individual value had been observed the most times? How many times was it observed?

State Information

A key consideration in simulating such an experiment is what information must be tracked. We should *not* need to store the full sequence of all selections thus far! We just need to maintain an appropriate "summary" of the results thus far.

3. What do you suggest would be the most natural (and minimal) information about past selections that should be maintained? Suggest a convenient way to represent that information in Python (keeping in mind that we want to write code assuming the number of coupons is a parameter, not necessarily 8).

Initialization

4. Give Python code that appropriately initializes your suggested data representation prior to the drawing of the first selection.

Update Rule

5. Assume that within the repetitive process we draw a new sample using the code

   `value = randrange(numberCoupons)`

   Describe using Python syntax how you would update your data structures in order to reflect the selection of the additional value.

Loop Condition

Our simulation needs to repeatedly process newly chosen random values, doing so until there is at least one occurrence of each of the available values.

6. If using a while loop with a style such as, **while not** done:, with some boolean variable done that is initially **False**. Give a snippet of code that could be used to reevaluate the setting for done after processing each successive random selection.

Empirical Results

While there are many ways to have implemented the above experiment, at the halfway point of this class, we will release our own (elegant) implementation on Moodle for download.

7. Run this source code five times using value $n = 100$. What do you observe about the number of selections needed for each trial before collecting all hundred "coupons"? What about the typical number of duplicates of a single coupon?

Multiple Trials

While we could have you run the above software over and over again, we may as well automate that process.

8. Modify the above code to ask the user for the number of experiment trials, and then have the code repeat the experiment that many times and reporting at the end the *average* number of numbers drawn for each individual experiment, as well as what was the *shortest* and what was the *longest* individual experiment in terms of number of selections. (No need to write your code here; work on a computer.)

   Report your findings for 10000 trials with $n = 100$.

Optimization (Bonus challenge)

If we try to further increase the size of n, this program becomes prohibitively slow. For example, simulating 10000 trials with $n = 1000$ required more than 3 minutes on our system. We attempted a trial with n equal to one million, but killed the unfinished experiment after 15 minutes. If required to do even one such experiment *by hand*, you would quickly realize there is an unnecessary inefficiency in our algorithm. After each individual selection, we potentially have to go through a long list of n counts to determine whether we have completed the full set.

A natural optimization is to maintain an additional variable, distinct, which keeps track of the number of different values that have been seen thus far in a trial. We can initialize it to zero and then our while condition can simply be rephrased as

```
while distinct < numCoupons:
```

Try implementing this optimization. Our implementation completed trials for $n = 1,000,000$ using about 28 seconds each. (Turns out you should expect to collect $1.42M$ coupons to get your complete set.)