The Most Common Topics Involving Randomization

This version of the document is dated 2020-08-06.

Peter Occil

Abstract: This article goes over some of the most common topics involving randomization in programming, and serves as a guide to programmers looking to solve their randomization problems. They were based on the most commonly pointed-to questions involving randomization on a Q&A site. The topics included generating uniform random numbers, unique random values, choosing one or more random items, shuffling, and querying random records from a database.

Introduction

This page goes over some of the most common topics involving randomization (including "random number generation") in programming, and serves as a guide to programmers looking to solve their randomization problems.

The topics on this page were chosen based on an analysis of the *Stack Overflow* questions that other questions were most often marked as duplicates of (using the *Stack Exchange Data Explorer* query named "Most popular duplicate targets by tag", with "random" as the TagName).

The analysis showed the following topics were among the most commonly asked:

- Generating uniform random integers in a range.
- Generating uniform random floating-point numbers in a range.
- Generating unique random integers in a range.
- Choosing a random item from a list.
- Choosing several unique items from a list.
- Choosing items with separate probabilities.
- Choosing random records from a database.
- Shuffling.
- Generating a random text string of characters selected from a restricted character set (such as only A to Z, a to z, 0 to 9).

Not all topics are covered above. Notably, the analysis ignores questions that were API-specific or programming-language specific, unless the underlying issue is present in multiple APIs or languages.

Another notable trend is that these topics were asked for programming languages where convenient APIs for these tasks were missing. This is why I recommend that **new programming language APIs** *provide functionality covering the topics above in their standard libraries*, to ease the burden of programmers using that language.

The following sections will detail the topics given above, with suggestions on how to solve them. Many of the links point to sections of my article "**Random Number Generation and Sampling Methods**".

The **pseudocode conventions** apply to this document.

All the randomization methods presented on this page assume we have a source of "truly" random and unbiased numbers.

Contents

Uniform Numbers in a Range

For algorithms on generating uniform random *integers* in a range, see **"Uniform Random Integers"** and "**A Note on Integer Generation Algorithms**" for a survey of algorithms. It should be noted there that most random number generators in common use output 32- or 64-bit non-negative integers, and for JavaScript, the idiom `(Math.random() < 0.5 ? 0 : 1)` will work in many practical cases as a random bit generator. Here is a JavaScript example of generating a random integer in the interval **`minInclusive`, `maxExclusive`), using the Fast Dice Roller by J. Lumbroso (2013)[**(1)**:

```
function randomInt(minInclusive, maxExclusive) {
  var maxInclusive = (maxExclusive - minInclusive) - 1
  if (minInclusive == maxInclusive) return minInclusive
  var x = 1
  var y = 0
  while(true) {
    x = x * 2
    var randomBit = (Math.random() < 0.5 ? 0 : 1)
    y = y * 2 + randomBit
    if(x > maxInclusive) {
      if (y <= maxInclusive) { return y + minInclusive }
      x = x - maxInclusive - 1
      y = y - maxInclusive - 1
    }
  }
}
```

Many common programming languages have no convenient or correct way to generate random numbers in a range. For example:

- Java's `java.util.Random` until version 8 had methods to produce `int`s in the interval [0, n) (`nextInt`), but not `long`s in that interval or integers in an arbitrary interval [a, b). Additional methods named `longs` and `ints` were later provided that offer this functionality, but even so, they are not as convenient in some cases than the existing `nextInt` method.
- JavaScript until recently has only one API for random number generation, namely `Math.random()`, and no built-in method for random integer generation or shuffling, among other things. Naïve solutions such as `Math.floor(Math.random()*x)+y` are not guaranteed to work reliably, in part because JavaScript doesn't require any particular implementation for `Math.random`.
- C's `rand` function produces random integers in a predetermined range ([0, `RAND_MAX`]) that is not within the application's control. This is just one of a **host of issues with `rand`**, by the way (unspecified algorithm, yet is initializable with "srand" for repeatability; non-thread-safety; unspecified distribution; historical implementations had weak low bits; etc.).

For algorithms on generating uniform random *floating-point numbers* in a range, see

"For Floating-Point Number Formats". Floating-point number generation has a number of issues not present with integer generation. For example, no computer can choose from all real numbers between two others, since there are infinitely many of them, and also, naïvely multiplying or dividing an integer by a constant (e.g., `Math.random()*x` in JavaScript) will necessarily miss many representable floating-point numbers (for details, see Goualard 2020[2]).

Choosing Random Items

In general, choosing a random item from a list is trivial: choose a random integer in `[0, n)`, where `n` is the size of the list, then take the item at the chosen position. The previous section already discussed how to generate a random integer.

However, if the number of items is not known in advance, then a technique called *reservoir sampling* can be used to choose one or more items at random. Here is how to implement reservoir sampling.

1. Set N to 1.
2. If no items remain, return the last chosen item. Otherwise, take the next item and choose it with probability 1/N.
3. Add 1 to N and go to step 2.

See "**Pseudocode for Random Sampling**" for an algorithm for reservoir sampling.

Unique Integers or Items

Generating unique random integers or items is also known as sampling *without replacement*, *without repetition*, or *without duplicates*.

There are many ways to generate unique items, depending on the number of items to choose, the number of items to choose *from*, and so on, and they have different tradeoffs in terms of time and memory requirements. See "**Sampling Without Replacement: Choosing Several Unique Items**" for advice.

Some applications require generating unique values that identify something, such as database records, user accounts, and so on. However, there are certain things to keep in mind when generating unique values for this purpose; see "**Unique Random Identifiers**" for more information.

Shuffling

An algorithm to randomize (*shuffle*) the order of a list is given in **"Shuffling"**. It should be noted that the algorithm is easy to implement incorrectly. Also, the choice of random number generator is important when it comes to shuffling; see my **RNG recommendation document on shuffling**.

Random Records from a Database

Querying random records (*rows*) from a database usually involves the database language SQL. However, SQL is implemented very differently in practice between database management systems (DBMSs), so that even trivial SQL statements are not guaranteed to work the same from one DBMS to another. Moreover, SQL has no loops, no branches, and no standard way to generate random numbers. Thus, the correct way to query random records from a database will vary from DBMS to DBMS.

With that said, the following specific situations tend to come up in random record queries.

- Querying one random record from a database.
- Querying a specified number of random records from a database.
- Querying one or more records each with a probability proportional to its weight. Very generally, this can be done by giving the table a column where each entry is a number generated as follows: `ln(R) / W` (where `W` is the record's weight greater than 0, and `R` is a per-record uniform random number in (0, 1)) (see also (Efraimidis 2015)[3]), then taking the records with the highest values of that column, but the efficiency of this technique depends on the DBMS.

Random Character Strings

Many applications need to generate a random string whose characters are chosen from a restricted set of characters. Popular choices include so-called *alphanumeric strings*, where the restricted character set is A to Z, a to z, 0 to 9. An algorithm for generating random strings is given in "**Random Character Strings**".

However, the following are some of the many considerations involving random string generation:

- If the string needs to be typed in by end users, or to be memorable, it may be important to choose a character set carefully or **allow typing mistakes to be detected**.
- If the string identifies something, the application may require strings it generates to be unique; see **Unique Random Identifiers** for considerations.
- If the string is in the nature of a password, a bearer credential, or another secret value, then it has to be generated using a **cryptographic RNG** (such as the `secrets` module in Python or the `random_bytes` function in PHP).

Choosing Items with Separate Probabilities

Weighted choice (also known as a *categorical distribution*) is a random choice of items, where each item has a *weight* and is chosen with a probability proportional to its weight. For algorithms on weighted choice, see "**Weighted Choice With Replacement**", which covers choices in which items are taken and put back.

The algorithm shown there is a straightforward way to implement weighted choice, but there are other alternatives (many of which are implemented in **Python sample code**). They include rejection sampling, Vose's version of the alias method (`VoseAlias`; see "**Darts, Dice, and Coins: Sampling from a Discrete Distribution**" by Keith Schwarz for more information), and the Fast Loaded Dice Roller (`FastLoadedDiceRoller`) (Saad et al. 2020)[4]. See "**A Note on Weighted Choice Algorithms**" for a survey of algorithms.

Weighted choice *without replacement* is a choice where each item can be chosen no more than once. The simplest way to implement this kind of weighted choice is to use weighted choice with replacement, except that after an index is chosen, that index's weight is set to 0 to keep the index from being chosen again. Other options are given in "**Weighted Choice Without Replacement (Single Copies)**" and "**Weighted Choice Without Replacement (List of Unknown Size)**".

Note that choosing *true* with a given probability, or *false* otherwise, is a special case of weighted sampling involving two items (also known as a *Bernoulli trial*). But there are much simpler ways of choosing *true* or *false* this way; see "**Boolean (True/False) Conditions**". Perhaps the most practical is the idiom `RNDINTEXC(Y) < X`, which chooses *true* with probability `X/Y`, *false* otherwise.

Other Topics

Other topics showed up in the analysis, and it's worth mentioning them here. These topics included:

- Generating a random *derangement*, or a random shuffle where every item moves to a different position (see **"Shuffling"**; see also `questions/25200220`).
- Generating a number that follows the **normal distribution**.
- Generating a number that follows an **arbitrary distribution**.
- **Random colors**.
- Random **numbers with a given sum**.
- Random **dates and times**.
- Stratified sampling (per-group sampling).
- Generating a **random point inside a circle**.

Notes

(1) Lumbroso, J., "**Optimal Discrete Uniform Generation from Coin Flips, and Applications**", arXiv:1304.1916 [cs.DS].

(2) Goualard, F., "**Generating Random Floating-Point Numbers by Dividing Integers: a Case Study**", 2020.

(3) Efraimidis, P. "**Weighted Random Sampling over Data Streams**", arXiv:1012.0256v2 [cs.DS], 2015.

(4) Saad, F.A., Freer C.E., et al. "The Fast Loaded Dice Roller: A Near-Optimal Exact Sampler for Discrete Probability Distributions", in *AISTATS 2020: Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research* 108, Palermo, Sicily, Italy, 2020.

License