Hat Random Access Container

Jonathan Wooldridge aka AngleWyrm
School of Hard Knocks, Bill Paying Sloggery and Amateur Hobbyist Entertainment, Seattle, USA
(Published October 31, 2004)

Herein is presented an algorithmic solution for generating random samples of a set within a time complexity of O(log n), along with an implementation for a container class template which performs this function. The container is an unsorted associative container, which uses probability weights for its key values. Also introduced is a paradigm for accessing data that gives rise to an intuitive usage of the phrase random access.

THERE are trivial methods available to create and use *uniform* random distributions, both with and without replacement. If one wishes to spin a wheel of outcomes, then it is a simple matter to scale a random number to the appropriate range, and select from an ordered set. If one wishes to draw cards from a deck, then it is also easy to perform a random_shuffle on the deck, and then take as many as desired from one end. Thus the uniform cases have elegant solutions, which perform in O(1) constant time.

The *non-uniform* cases are a different matter. In 1977, A.J. Walker [1] provided a solution to weighted sampling without replacement, with an alias technique, which involves the generation and use of a clever but non-intuitive lookup table. In 1997, Donald Knuth [2] presented an implementation of this technique, which performs in a very efficient constant time, once a lookup table has been created, which requires linear time.

But there is still one class of problem unresolved. In situations where items are dynamically added to or removed from the set, the above alias technique must recalculate the lookup table for each insert/delete, a linear time operation. Examples of this problem space are drawing lottery balls, or randomly selecting from a changing client list. Neither does it address situations where the probability weights of the items change during run-time. This is scenario is found in AI learning and categorization algorithms.

Orientation

Let an experiment consist of a set S of k possible outcomes $s_i \in S$, having associated probability weights $P(s_i)$ such that:

$$\sum_{i=1}^{k} P(s_i) = 1 \quad ; \quad 0 \prec P(s_i) \leq 1 \qquad (1)$$

Note in particular that $P(s_i) > 0$ for all s_i; an impossible outcome is not a member of the set of possible outcomes.

For reason of algorithmic optimization let us express $P(s_i)$ as a numerator/denominator integer pair:

$$P(s_i) = \frac{c_i}{\sum_{j=1}^{k} c_j} \quad ; \quad c_i \succ 0 \qquad (2)$$

In equation (2), c_i can be interpreted as the number of chances out of $m = \sum c_i$ total chances that outcome s_i will result from experiment S. This value c_i can be defined as the *raw chance weight* of s_i with respect to probability, which can then be viewed as the normalized chance weight of s_i.

With these raw chance weights, we can now completely express the probability distribution of the k outcomes of experiment S with $k+1$ integers k, $C=\{ c_1, c_2, \ldots, c_{k-1}, c_k \}$, where the set C of raw chance weights $c_i \in C$ constitute a partition of the integer $m = \sum c_i$ of width k.

The Container

Given a random variable x with $0 \quad x \quad m$, we wish to return an outcome $s_i \in S$, with probability proportional to the associated raw chance weight $c_i \in C$.

We can accomplish this by maintaining a binary tree of cumulative chance weights.

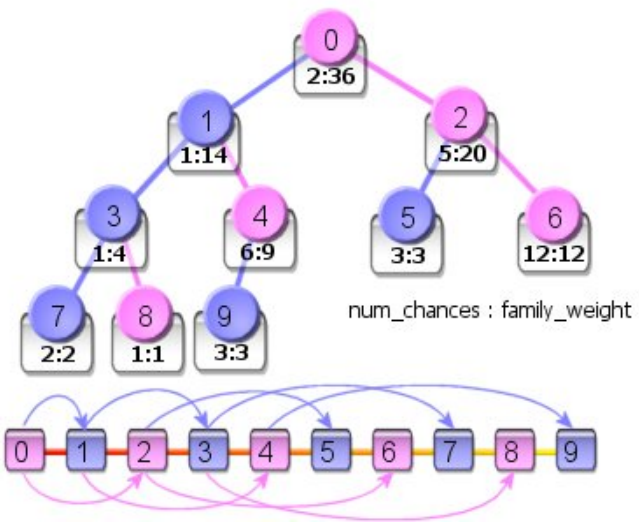

Figure 1: Cumulative chance weights stored in nodes

In figure (1), the variables on the right side of each node, listed as family_weight, refer to the recursive accumulation of the node's chance weight c_i (listed on the left side of each node), plus the recursive sum of the node's two children family_weight values.