ELSEVIER

Information Processing Letters 70 (1999) 255–257

Information Processing Letters

www.elsevier.com/locate/ipl

In-place random list permutations

Wen-Ping Hwang [a,*], Ching-Lin Wang [b,1]

[a] *Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan*
[b] *Department of Computer Science and Information Engineering, Tamkang University, Tamsui, Taiwan*

Received 10 February 1999; received in revised form 10 June 1999
Communicated by K. Iwama

Abstract

We give two algorithms to randomly permute a linked list of length n in place using $O(n \log n)$ time and $O(\log n)$ stack space in both the expected case and the worst case. The first algorithm uses well-known sequential random sampling, and the second uses inverted sequential random sampling.

Keywords: Algorithms; Data structures

1. Classic algorithm

Consider the problem of permuting a list uniformly at random. The classic swapping algorithm [1] for randomly permuting an array can be adopted to solve this problem: Randomly choose an element out of a list of length n as the last element of the permuted list, and recursively permute the remaining list of length $n-1$ in a like manner to form the first $n-1$ elements of the permuted list.

Due to the need of traversing the list to locate the randomly chosen element, the algorithm runs in $O(n^2)$ time and $O(1)$ auxiliary space in both the expected case and the worst case.

2. Ressler's algorithm

Like the classic algorithm, Ressler's algorithm [2] first randomly chooses an element out of a list of length n as the last element of the permuted list. It then splits the remaining $n-1$ elements into two sublists according to a sequence of $n-1$ coin tosses. The two sublists are then recursively permuted in a like manner, and the resulting two permuted sublists are appended to form the first $n-1$ elements of the permuted list.

Much like the randomized quicksort algorithm, Ressler's algorithm takes $O(n \log n)$ time and $O(\log n)$ stack space in the expected case. But in the worst case, it takes $O(n^2)$ time and stack space.

3. Random permutations with random sampling

Our first algorithm makes use of sequential random sampling. Given a list of length $n > 1$, select a random sample of $\lfloor n/2 \rfloor$ elements and place the selected elements in one sublist and the rejected elements in another. Recursively permute the two sublists in a like manner and then append the two permuted sublists into a permuted list.

* Corresponding author. Email: wphwang@cis.nctu.edu.tw.
[1] Email: clwang@cs.tku.edu.tw.

0020-0190/99/$ – see front matter
PII: S0020-0190(99)00076-9

It is easily seen by induction that the algorithm produces each of $n!$ permutations with equal probability. For $n \leqslant 1$, the result is obvious. Suppose that the algorithm produces a random permutation for any list of length less than n. Then any permutation of a list of length n is obtained with probability

$$\frac{1}{\binom{n}{\lfloor n/2 \rfloor}} \cdot \frac{1}{\lfloor n/2 \rfloor!} \cdot \frac{1}{\lceil n/2 \rceil!} = \frac{1}{n!},$$

where the first term is the probability of obtaining an $\lfloor n/2 \rfloor$-combination of the list and the second and the third terms are assured by the induction hypothesis.

The random sampling step can be done in $O(n)$ time in both the expected case and the worst case using Algorithm S of Knuth [1, p. 142]. The concatenation of the two permuted sublists can be done in $O(n)$ time. Alternatively, the concatenation step can be eliminated by the standard technique of using an accumulating parameter to accumulate the required permuted list [2]. It follows that the algorithm takes $O(n \log n)$ time in both the expected case and the worst case. As for the space complexity, it is easily seen that $O(\log n)$ stack space is required.

4. Random permutations with inverted random sampling

Central to our first algorithm is the fact that any combination of m elements of a list has equal probability of occupying the first m positions, regardless of the order, of a random permutation of the list. Conversely, any combination of m positions of a random permutation of a list has equal probability of being occupied by the first m elements, regardless of the order, of the list. This observation leads to the next algorithm that uses 'inverted' sequential random sampling.

A sequential random sampling procedure produces a random sample sequence and a rejected sequence from a population sequence. On the other hand, an inverted sequential random sampling procedure produces a random population sequence from a sample sequence and a rejected sequence. Intuitively, an inverted sequential random sampling procedure produces, with equal probability, any of the population sequences from which the corresponding sequential random sampling procedure may yield the sample and rejected sequences, and *vice versa*.

As an example, Algorithm S of Knuth, the best-known sequential random sampling algorithm, can be inverted by undoing the selection and rejection operations, as shown in Algorithm S^{-1}.

Algorithm S^{-1}. /* Compute a random population sequence from a sample sequence of m items and a rejected sequence of $n-m$ items, where $0 < m \leqslant n$. */

1. [Generate U] Generate a random number U that is uniformly distributed between 0 and 1.
2. [Test] If $nU \geqslant m$, go to step 4.
3. [Unselect] Select the next item from the sample sequence for the population, and decrease n and m by 1. If $m > 0$, go to step 1; otherwise,
   [Unreject] select each of the remaining items, if any, in the rejected sequence for the population, and then terminate the algorithm.
4. [Unreject] Select the next item from the rejected sequence for the population, decrease n by 1, and go to step 1.

It is not hard to see that Algorithms S and S^{-1} are inverse to each other in the sense that

$$\begin{aligned} &P\big[S^{-1}\big(S(p_n, m)\big) = p_n\big] \\ &\quad = P\big[S\big(S^{-1}(s_m, r_{n-m}), m\big) = (s_m, r_{n-m})\big] \\ &\quad = 1/\binom{n}{m}, \end{aligned}$$

where $S(p_n, m)$ is the pair of the m-element sample sequence and the $(n-m)$-element rejected sequence produced by Algorithm S on the n-element population sequence p_n, and $S^{-1}(s_m, r_{n-m})$ is the n-element population sequence produced by Algorithm S^{-1} on the pair of the m-element sample sequence s_m and the $(n-m)$-element rejected sequence r_{n-m}. Furthermore, like Algorithm S, Algorithm S^{-1} produces an unbiased population sequence with probability $1/\binom{n}{m}$ and runs in $O(n)$ time in both the expected case and the worst case.

Our second algorithm is as follows. Given a list of length $n > 1$, split it at the middle to obtain two sublists. Recursively permute the two sublists in a like manner. Treat the two permuted sublists as sample and rejected sequences, respectively, and apply an inverted sequential random sampling procedure to obtain a

random population sequence, which is the desired permuted list.

It is easily seen by induction that this algorithm produces any permutation of the list unbiasedly at random. Furthermore, the algorithm undoubtedly runs in $O(n \log n)$ time and $O(\log n)$ stack space in both the expected case and the worst case.

References

[1] D.E. Knuth, The Art of Computer Programming, Vol. 2: Semi-Numerical Algorithms, 3rd edn., Addison-Wesley, Reading, MA, 1998.

[2] E.K. Ressler, Random list permutations in place, Inform. Process. Lett. 43 (1992) 271–275.