(CE7456 Lecture Notes 1) Weighted Sampling

Yufei Tao

Today, we will discuss the *weighted sampling* problem defined as follows.

> **Weighted Sampling.** The input is a set S of n elements where each element $e \in S$ carries a positive integer *weight* $w(e)$. Define $W = \sum_{e \in S} w(e)$. A sampling operation draws a random element X from S such that $\mathbf{Pr}[X = e] = w(e)/W$ for each $e \in S$. The goal is to preprocess S into a data structure that can support sampling operations efficiently. The output of each sampling operation must be independent of those of all previous operations.

1 Computation Model and Mathematical Conventions

We will assume the standard RAM (random access machine) model, augmented with a constant-time operation `RAND`. Given an integer $x \geq 1$, `RAND`(x) returns a number chosen from $[x]$ uniformly at random, where $[x]$ represents the set $\{1, 2, ..., x\}$. All the logarithms have base 2 by default.

2 A Simple but Slow Method

Let $e_1, e_2, ..., e_n$ be the elements of S (the ordering does not matter). In preprocessing, break the interval $[1, W]$ into n disjoint intervals $I_1, I_2, ..., I_n$ such that I_i ($i \in [n]$) contains exactly $w(e_i)$ integers. Store the set of intervals in an array A, which constitutes our data structure. It is rudimentary to finish the preprocessing in $O(n)$ time.

To sample from S, first draw an integer X from $[n]$ uniformly at random. Then, use binary search to identify the unique interval I_i — among $I_1, I_2, ..., I_n$ — covering X. Finally, return the element e_i (i.e., the element I_i corresponds to). The correctness is straightforward, and the sample time is $O(\log n)$.

3 The Alias Method

Next, we will discuss the *alias method* [2], which improves the above solution by reducing the sampling time to constant.

Structure. The method produces a set U of n *urns*, each containing one or two elements from S. An element may appear in multiple urns. For an urn $\Lambda \in U$, each element e in Λ is assigned a positive value $v(\Lambda, e)$ — referred to as *the value of* e *in urn* Λ. These values satisfy two conditions:

1. For each urn $\Lambda \in U$, if it has only one element e, then $v(\Lambda, e) = W/n$; otherwise, it has two elements e_1 and e_2, in which case $v(\Lambda, e_1) + v(\Lambda, e_2) = W/n$.

2. For every element $e \in S$, it holds that

$$w(e) \quad = \sum_{\Lambda \in U : e \in \Lambda} v(\Lambda, e) \tag{1}$$

namely, its weight equals the sum of its values in all the urns where it appears.

As each urn requires only constant space, the total space consumption is $O(n)$.

Sampling. To perform a weighted sampling operation, we carry out the steps below.

- First, pick an urn Λ from U uniformly at random.
- Second, return an element X from U as follows.
  - If Λ has only a single element e, then $X = e$.
  - If Λ has two elements e_1 and e_2, then generate $X \in \{e_1, e_2\}$ such that

$$\mathbf{Pr}[X = e_i] \quad = \quad \frac{v(\Lambda, e_i)}{W/n}$$

    for $i = 1$ and 2.

  We insert a remark here that will be useful later. If e is an element in Λ, in both cases we have $\mathbf{Pr}[X = e] = v(\Lambda, e) \cdot (n/W)$.

It is clear that the above steps can be implemented in constant time.

To prove correctness, denote by Y the output of the algorithm; we must show that $\mathbf{Pr}[Y = e] = w(e)/W$ for each element $e \in S$. Fix an arbitrary element $e \in S$. Let U_e be the set of urns in U containing e. The event $Y = e$ can be decomposed into $|U_e|$ disjoint events $E_1, E_2, ..., E_{|U_e|}$, where E_i ($1 \le i \le |U_e|$) represents the event that e is sampled from the i-th urn of U_e. Specifically, E_i ($1 \le i \le |U_e|$) is the conjunction (i.e., AND) of the following two events:

- The urn Λ picked in the first step is the i-th urn in U_e, which occurs with probability $1/n$.
- The element X returned in the second step is e, which occurs with probability $v(\Lambda, e) \cdot (n/W)$ (as remarked earlier).

As the above two events are independent, we know $\mathbf{Pr}[E_i] = v(\Lambda, e)/W$. Therefore:

$$\mathbf{Pr}[Y = e] = \sum_{i=1}^{|U_e|} \mathbf{Pr}[E_i] = \sum_{\Lambda \in U_e} \frac{v(\Lambda, e)}{W} = \frac{w(e)}{W}$$

where the last equality used (1).

Construction. We can build the urns in n iterations. Each iteration produces a new urn, removes an element from S, and possibly adjusts the weight of another element remaining in S. The algorithm maintains an invariant.

> **Invariant:** when step $i \in [1, n]$ starts, the weights of all the $n - i + 1$ elements still in S sum up to $W \cdot \frac{n-i+1}{n}$.

Specifically, in the i-th iteration ($i \ge 1$), we first check whether there is an element $e \in S$ whose *current* weight is W/n. If so, the iteration creates an urn Λ containing just e, assigns $v(\Lambda, e) = w(e)$ — here $w(e)$ is the current weight of e — and then removes e from S.

We now consider the case where no element in S has a current weight W/n. In this case, there must be an element $e_1 \in S$ whose current weight $w(e_1)$ is *strictly* smaller than W/n, and another element $e_2 \in S$ whose current weight $w(e_1)$ is *strictly* larger than W/n — think: why?

Pick any two such elements. Create an urn Λ containing e_1 and e_2, assigning $v(\Lambda, e_1) = w(e_1)$ and $v(\Lambda, e_2) = \frac{W}{n} - w(e_1)$. After that, the iteration removes e_1 from S and decreases $w(e_2)$ by $\frac{W}{n} - w(e_1)$. It is easy to verify that the invariant holds for the next iteration.

The above algorithm can be implemented in $O(n)$ time. The details make an interesting exercise for you (hint: maintain three linked lists).

4 Remark

It is possible to update the alias structure in constant time [1] when inserting a new element in S or deleting an existing element from S.

References

[1] Torben Hagerup, Kurt Mehlhorn, and J. Ian Munro. Maintaining discrete probability distributions optimally. In *ICALP*, pages 253–264, 1993.

[2] Alastair J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974.