APPM 4/5560 - Laboratory 1 - Fall 2016

by: Manuel Lladser

Instructions: This lab is due at the beginning of lecture on Monday September 26 2016. As a recommendation, concentrate first on the theoretical questions and just then implement the pseudo-codes in a computer to perform simulations.

Students registered for APPM 4560 should work in groups of 2-3 members and submit one lab report with all participant names on it. All participants should be equally responsible and involved in the lab assignment. Students registered for APPM 5560 must solve the lab on their own.

By submitting a report, all its participants agree to comply with the CU Honor Code Policy.

Your report is limited to 5 pages with a minimum font size of 11 points and 1 inch margins; in particular, please provide complete but brief answers. Do not include an introduction nor a conclusion section. To receive full credit you must submit a professional report addressing all the instructions and questions in the same order as requested. Be sure to include all figures or tables and to label each of them (e.g. Figure 1, Table 2, etc.). Also be sure to include all pseudo-codes requested. Pseudo-codes should follow the same format used in class, and should relay only on the simulation of uniform random variables on the interval $(0, 1)$. Your write-up should include brief but complete answers to all the questions listed below with appropriate references to labeled figures or tables. Good luck!

Part I: Simulating Random Permutations

In what follows $n \geq 1$ is a given integer. A *permutation* of the elements in the set $\{1, \ldots, n\}$ is any ordered list of these elements where no element appears repeated. Examples of three different permutations when $n = 3$ are $(1, 2, 3)$, $(3, 1, 2)$ and $(2, 1, 3)$. In the last permutation, 2 is the first element, 1 the second, and 3 the third. Observe that according to the multiplicative principle there are $(3 \cdot 2 \cdot 1)$ i.e. 3! permutations of the elements in the set $\{1, 2, 3\}$.

More generally, there are $n!$ different permutations of the elements in the set $\{1, \ldots, n\}$. If we chose any of these at random and without preference for one permutation more than another, we obtain a *random permutation*. In particular, any specific permutation has a chance of $1/n!$ to be selected. For example, if $n = 3$ then each of the permutations $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$, $(3, 2, 1)$ has 1 out of 6 chances to be selected.

Here is a simple way to generate a random permutation of $\{1, \ldots, n\}$: simulate i.i.d. $Uniform(0, 1)$ random variables $U_1, \ldots, U_n$, and associate with these the (random) function $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ defined as $f(i) = \#\{j : U_j \leq U_i\}$. The random permutation is then $\sigma := \big(f(1), \ldots, f(n)\big)$. For example, if $n = 3$ and $U_1 = 0.0321...$, $U_2 = 0.4740...$, $U_3 = 0.2298...$ then $\sigma = (1, 3, 2)$. Instead, if $U_1 = 0.9455...$, $U_2 = 0.3221...$, $U_3 = 0.1001...$ then $\sigma = (3, 2, 1)$.

(1.1) Use the above to design an algorithm that produces a random permutation. The input of the algorithm is an integer $n \geq 1$ and the output is a random permutation of the elements in the set $\{1, \ldots, n\}$. To design the algorithm assume you can simulate any number of independent uniform random variables in the interval $(0, 1)$.

In what remains of this section $n = 7$, $\sigma = (6, 7, 2, 5, 1, 4, 3)$ and $m = 6000$.

(1.2) Let X be the number of times that the permutation σ is observed in m runs of your algorithm. What's the distribution of X? What's the expected value of X? Explain.

(1.3) Let Y be the random variable that counts the number of times you need to run your algorithm until seeing the permutation σ for the first time (instructions within a single run of the algorithm do not account for Y). What's the distribution of Y? What's the expected value of Y? Explain.

Implement your algorithm in a computer. In what remains of this section $k = 2000$.

(1.4) Use your algorithm to obtain k independent realizations of X and obtain the histogram associated with these. How does the resulting histogram compare to the theoretical histogram of X? Explain. Display the histogram and the theoretical distribution on the same plot. Make sure to comment on any expected or unexpected behavior.

(1.5) Use your algorithm to obtain k independent realizations of Y and obtain the histogram associated with these. How does the resulting histogram compare to the theoretical histogram of Y? Explain. Display the histogram and the theoretical distribution on the same plot. Make sure to comment on any expected or unexpected behavior.

Part II: Comparing Performance of Retrieving Algorithms

Imagine you have an oracle that always answers the truth to a YES/NO question. Suppose also that you have a database with n entries and that your goal is to retrieve a special entry by asking questions to the oracle. To simplify things, we will assume that the database is a permutation of the elements $\{1, \ldots, n\}$ and that you want to retrieve the position where the element 1 is located. For example, if $n = 3$ and the permutation is $(2, 3, 1)$ then 1 is the third entry.

Here are two methods to retrieve the position 1.

Method A. You ask the oracle: is 1 in the first position?, is 1 in the second position?, etc until the oracle answers YES.

To define the second method recall that $\lfloor x \rfloor$, called the *integer part of x*, is defined as the largest integer that is less or equal to x. For example, $\lfloor 1.23 \rfloor = 1$ and $\lfloor 0.9999 \rfloor = 0$.

Method B. You first ask the oracle: is 1 in the first $\lfloor n/2 \rfloor$ positions? If the answer is YES then your next question would be: is 1 in the first $\lfloor n/4 \rfloor$ positions? However, if the answer was NO then your next question would be: is 1 in the last $n - \lfloor n/2 \rfloor$ positions i.e. between positions $1 + \lfloor n/2 \rfloor$ and n? You keep applying this method recursively until the oracle responds YES.

For example, if $n = 16$ and the permutation was $(5, 14, 13, 9, 3, 12, 16, 6, 7, 2, 11, 1, 4, 10, 8, 15)$ then, using the Method A, you would need twelve questions to determine that 1 was at position twelve. On the other hand, using the method B, with the first question you would eliminate positions 1–8, with the second question positions 13–16, with the third question positions 9–10, and your fourth question would be: is 1 at position 11? The oracle would respond NO and you would conclude that 1 was at position twelve. Hence, using the method B you would only ask four questions to find the position of 1.

In what follows Q_A will denote the total number of questions made to the oracle to determine the position of 1 using the Method A. We define Q_B in analogous manner. In real databases data is inserted or removed constantly. Hence it is reasonable to assume that after several insertions and deletions any particular key one would like to retrieve may be located anywhere in the database. In terms of our discussion above, this motivates to model the permutation where we will search for 1 to be a random permutation. This makes Q_A and Q_B random variables.

It should be intuitively clear that—in average—Method B is much more efficient than the Method A when n is large. In this part of the lab you will support this intuition with the aid of simulations.

(2.1) Determine $\mathbb{E}(Q_A)$ explicitly. Justify mathematically!

(2.2) Fix $n = 9$ and use your algorithm in the first part to simulate ten-thousand times the random variable Q_A. Determine the sample average of your simulations. Repeat the same instructions with $n = 21$, $n = 36$ and $n = 69$.

HINT: You do not want to simulate all the ten-thousand permutations before determining Q_A in each. Furthermore, you do not need to keep in memory the random permutation you are generating to determine the location of 1 at the very end. Modify your algorithm in part (1.1) so that you only keep track of the position of 1 in the permutation as you build it using the pairs (i, U_i).

(2.3) How do the averages compare to your answer in part (2.1)? Summarize your data in a table. Comment on any expected and/or unexpected behavior.

(2.4) What should $\mathbb{E}(Q_B)$ be approximately when n is large? (An intuitive justification suffices!)

(2.5) Repeat the instructions in (2.2) above but with Q_B instead of Q_A. How do the obtained averages compare to your approximation in part (2.4)? Summarize your data in a table and comment on any expected or unexpected behavior.

Final Comments.

You may ask: what does all this have to do with retrieving algorithms?

The Method A models the most natural method to retrieve data from a database. On the other hand, the Method B is a simplistic model for a data structure called *binary search tree.* A *binary tree* is an acyclic graph in which every node has degree at most three i.e. it is connected through an edge to at most three other nodes. The tree is said to be *rooted* when there is a marked node (called the *root*) of degree at most two. In a rooted binary tree there is a unique path of shortest length connecting any node with the root. This allows to talk about nodes at distance zero, one, two, etc from the root. A node u is said to be a *parent* of v, equivalently v is said to be a *descendant* of u, if u and v are connected however $distance(v, root) = 1 + distance(u, root)$.

If you were to represent data using bits e.g. 001, 010, 101, and 111 then the data could be "stored" in a binary search tree like the figure below. To store a new data in the tree we would read it from left to right and, starting at the root, we would descent to the left every time we read a 0, and to the right every time we read a 1. This way of storing data in the tree allows you for an efficient retrieval: if the tree is well-balanced then every time we descend down in it searching for a key we will eliminate about half of the remaining data that we are not interested in retrieving!

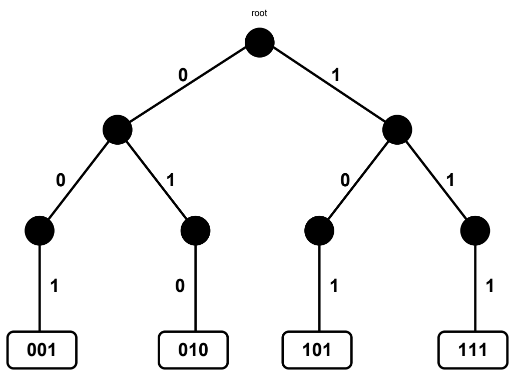

Figure 1: Example of a well-balanced binary search tree storing the keys 001, 010, 101, and 111.