CS466 — Fall 1999

Problem Set 2

Assigned problems

These problems are due Monday, October 25 (details to be announced).
Do your own work.

1. CLR problem 21-1, p. 438.
2. BB problem 10.36, pp. 372–3.
3. (a) Give an efficient algorithm to merge two skip lists, the first having m items and maximum node height g and the second having n items and maximum node height h.

   (b) Show that your algorithm has an average-case run time in $O(m+n)$ (averaging over the structure of the input lists as well as any random choices made during the merge). What is the worst-case run time?

   (c) Compare the properties of skip lists to those of treaps. Discuss when each might be used appropriately. Include specific applications if possible.

(Problem 4 on the next page.)

4. Alex and Sam are writing a program that needs to generate a random permutation of the integers from 1 to n in an array P.

Alex: We can select random values to put into P easily, but we need to make sure we don't get the same value twice. We could check for duplicates, and chose again if we get one. But that would take a long time to find the last few—most of the choices would be duplicates.

Sam: Hmm. Perhaps we can start with the values 1 to n in the array, and move them around at random. Something like this:

For $i \leftarrow 1$ to n
    $P[i] \leftarrow i$
For $i \leftarrow 1$ to n
    $j \leftarrow \text{RANDINT}(1, n)$
    exchange $P[i]$ and $P[j]$

(My function RANDINT(1,n) chooses a number uniformly between 1 and n.) Then we won't have to worry about duplicates, since the values in P are always distinct.

Alex: Interesting! But I wonder about the probabilities—does it give all permutations with equal probability? Hmm....

Uh, oh, something's wrong. The n calls to RANDINT give n^n different executions of the algorithm. Thus we can't possibly get each permutation with equal probability—$n!$ doesn't divide n^n evenly.

(a) Give an explicit example of two permutations of size n that are generated with different probabilities by the algorithm above. (Hint: choose a small value of n.)

(b) Modify the algorithm to generate each permutation with equal probability, assuming that RANDINT() correctly generates a uniform distribution on $\{1, 2, \ldots, n\}$. Prove that your algorithm is correct.

(c) Suppose that RANDINT(1,n) uses a pseudo-random generator that returns a binary string (as the standard linear-congruential ones do). Thus if n is not a power of 2, a single call to the pseudo-random generator cannot give exactly the required probability. Discuss how this property affects your algorithm. How would you use a pseudo-random generator to get the best results?