OLD CS 473: Fundamental Algorithms, Spring 2015

Discussion 9

March 11, 2015

9.1. Matrix verification.

You are given three matrices A, B, C of positive integer numbers. The matrices are each of size $n \times n$. The claim is that $AB = C$. To verify this, your algorithm randomly chooses a random vector $r = (r_1, r_2, \ldots, r_n) \in \{0,1\}^n$, and computes $x = ABr$ by computing $A(Br)$, which can be done in $O(n^2)$ time (how?). It then computes $y = Cr$. The algorithm returns that "$AB = C$" if $x = y$ and "$AB \neq C$" if $x \neq y$. Prove, that given that $AB \neq C$ then the probability that the algorithm returns "$AB \neq C$" is at least half.

9.2. Lossy hashing.

Consider the problem where given m elements $t_1, \ldots, t_n$, you are going to hash them in an array of size n. Assume you are given a random hash function that is perfectly uniform; that is, for every element you have uniform distribution of where the element might be mapped to in the array. If two or more elements gets mapped to the same entry in the array, we throw them away.

(A) What is the probability of an element to survive; that is, to be the only one mapped to its cell in the array?

(B) A pair of elements ***collides*** if the two elements are being hashed to the same cell in the array by h. What is the total expected number of colliding pairs?

(C) What is the probability that no pair of elements collide?

9.3. Find kth smallest number.

This question asks you to design and analyze a *randomized incremental* algorithm to select the kth smallest element from a given set of n elements (from a universe with a linear order). In an *incremental* algorithm, the input consists of a sequence of elements $x_1, x_2, \ldots, x_n$. After any prefix $x_1, \ldots, x_{i-1}$ has been considered, the algorithm has computed the kth smallest element in $x_1, \ldots, x_{i-1}$ (which is undefined if $i \leq k$), or if appropriate, some other invariant from which the kth smallest element could be determined. This invariant is updated as the next element x_i is considered.

Any incremental algorithm can be *randomized* by first randomly permuting the input sequence, with each permutation equally likely.

(A) Describe an incremental algorithm for computing the kth smallest element.

(B) How many comparisons does your algorithm perform in the worst case?

(C) What is the expected number (over all permutations) of comparisons performed by the randomized version of your algorithm? (Hint: When considering x_i, what is the probability that x_i is smaller than the kth smallest so far?) You should aim for a bound of at most $n + O(k \log(n/k))$. Revise (a) if necessary in order to achieve this.

[Notice, that if k is much smaller than n, then you algorithm performs $n + o(n)$ comparisons.]

9.4. Sorting Random Numbers

Suppose we pick a real number x_i at random (uniformly) from the unit interval, for $i = 1, \ldots, n$. Our purpose is the following (which we are not going to achieve in this question).

> Describe an algorithm with an expected linear running time that sorts $x_1, \ldots, x_n$.

To make this question more interesting, assume that we are going to use some standard sorting algorithm instead (say merge sort), which compares the numbers directly. The binary representation of each x_i can be generated as a potentially infinite series of bits that are the outcome of unbiased coin flips. The idea is to generate only as many bits in this sequence as is necessary for resolving comparisons between different numbers as we sort them. Suppose we have only generated some prefixes of the binary representations of the numbers. Now, when comparing two numbers x_i and x_j, if their current partial binary representation can resolve the comparison, then we are done. Otherwise, the have the same partial binary representations (upto the length of the shorter of the two) and we keep generating more bits for each until they first differ.

(A) Compute a tight upper bound on the expected number of coin flips or random bits needed for a single comparison.

(B) Generating bits one at a time like this is probably a bad idea in practice. Give a more practical scheme that generates the numbers in advance, using a small number of random bits, given an upper bound n on the input size. Describe a scheme that works correctly with probability $\geq 1 - n^{-c}$, where c is a prespecified constant.

9.5. What happens on Uranus stays on Uranus.

There are currently k Uranusians alive on Uranus (they look a lot like cucumbers, but with legs). Each month exactly one of the Uranusians undergoes a critical event. Either it dies with probability p or it splits into two new Uranusians with probability p. With probability $1 - 2p$ nothing happens.

(A) Let X_i be the size of the population of the Uranus population after the ith month. What is $\mathbf{E}[X_i]$? Here, you need to only provide a good upper bound on $\mathbf{V}[X_i]$, as computing it exactly seems hard.

(B) Let P_i be the probability that the population of Uranus is non-zero after i months. Prove that $\lim_{i=0}^{\infty} P_i = 0$. (There is a short and elegant argument showing that, but it is in fact not easy to see.)

(C) (Harder.) Let X be the number of months till the population of Uranus is extinct. Prove that $\mathbf{E}[X]$ is unbounded.

This is a very simple example of a Galton-Watson process. From wikipedia:

> The Galton-Watson process is a branching stochastic process arising from Francis Galton's statistical investigation of the extinction of family names.
>
> There was concern amongst the Victorians that aristocratic surnames were becoming extinct. Galton originally posed the question regarding the probability of such an event.