6.856
Randomized Algorithms
2005/03/09

Dan Ports
drkp@mit.edu
Collaborators: {sarahl, gpickard}

Problem Set 5

Problem 1:

a) Suppose the n bits are divided into r blocks of size n/r, and each block has its own 2-universal hash function. Assume that at most m items have been inserted into the data structure, and we are adding a new item x. Since there are m other items and n/r spaces, the probability that some other item hashes to the same value as x in one block is $\frac{m}{n/r} = \frac{mr}{n}$. This requires only 2-universality of the hash function since we consider only the pairwise comparison of whether some other element y hashes to the same values as x.

The probability that there is a false positive for an element x is the probability that some other item hashes to the same value as x in each of the r blocks. This is

$$\left(\frac{mr}{n}\right)^r$$

b) To find the minimum of this equation, we set the derivative equal to zero. For simplicity, let $k = m/n$. Then we are differentiating $(kr)^r$. Applying advanced calculus techniques[1], we find that

$$\frac{d}{dr}(kr)^r = (\ln kr + 1)(kr)^r = 0$$
$$\ln kr = -1$$
$$r = \frac{1}{ke}$$

So to minimize the probability of a false positive, we choose $r = \frac{1}{ke} = \frac{n}{me}$.

Problem 2:

Suppose sets X and Y each have m elements, with r elements in common. We consider the expected number of bits where two n-bit Bloom filters differ, using the same k random hash functions.

Consider first the elements that X and Y have in common. There are r such elements, so the probability that a given bit is clear is the probability that all k hash functions for all r elements map to some other bit, or $(1 - 1/n)^{kr}$. Considering the indicator random variable for whether each bit is clear, summing, and applying linearity of expectation, the expected number of bits left clear from the first r is

$$\mathbf{E}[\#\text{ bits clear}] = n\left(1 - \frac{1}{n}\right)^{kr}$$

This is the number of bits that can differ between the two sets' Bloom filters; we know that the other bits will be the same because the r elements in common will hash to the same values in each set and set those bits to 1.

So consider one of the bits that is clear after inserting the r common elements into the Bloom filter. We will bound the probability that this bit differs in the two Bloom filters. The probability

[1] Involving computer algebra systems and enslaved freshmen who haven't forgotten their calculus yet.

that the bit is set in one of the Bloom filters is $\left(1-\frac{1}{n}\right)^{k(m-r)}$. Call this probability p. Then the probability that this bit differs in the two Bloom filters is $2p(1-p)$, or the probability that it is set in exactly one of the two filters. So

$$\mathbf{Pr}\left[\text{bit differs}\right] = 2\left(1-\left(1-\frac{1}{n}\right)^{k(m-r)}\right)\left(1-\frac{1}{n}\right)^{k(m-r)}$$

Writing this as indicator random variables for each of the variables that are not set by the r common elements and applying linearity of expectation, we find that

$$\begin{aligned}\mathbf{E}\left[\#\text{ differing bits}\right] &= n\left(1-\frac{1}{n}\right)^{kr}\left(2\left(1-\left(1-\frac{1}{n}\right)^{k(m-r)}\right)\left(1-\frac{1}{n}\right)^{k(m-r)}\right)\\ &= 2n\left(1-\left(1-\frac{1}{n}\right)^{k(m-r)}\right)\left(1-\frac{1}{n}\right)^{km}\end{aligned}$$

Note that this is a monotonic function with respect to r. Thus, to estimate the number of elements that differ between two sets X and Y, we can compute Bloom filters over each set, exchange the Bloom filters, and determine the number of bits that differ. We can then compute the value of r that gives, in expectation, the observed number of differing bits. This is an estimate for the number of elements in common between the two sets.

Problem 3:

To test the isomorphism between two trees T_1 and T_2, we reduce each tree to a polynomial and check the equality of the polynomials using Freivalds' technique. Following the hint, we define a multivariate polynomial P_v for each vertex v, where $P_v = x_0$ for a leaf vertex, and for a vertex v of height h with children $v_1, \ldots v_k$, $P_v = (x_h - P_{v_1}) \cdots (x_h - P_{v_k})$. The polynomial associated with the tree is P_r, where r is the root of the tree.

Note first that equality of the polynomials associated with two trees is both necessary and sufficient to guarantee the isomorphism of the trees. If two trees are isomorphic, they will have the same polynomial: for each vertex v of height h with children $v_1, \ldots v_k$, since the tree structures are identical up to the vertex names and ordering of children,

$$P_v = (x_h - P_{v_1}) \cdots (x_h - P_{v_k}) = (x_h - P_{f(v_1)}) \cdots (x_h - P_{f(v_k)}) = P_{f(v)}$$

since the ordering of product terms in the polynomial does not matter. Moreover, if two trees have the same polynomial, they are isomorphic. To see this, note that the tree can be uniquely reconstructed (up to ordering and naming of vertices) from the polynomial. We can do so by identifying the height h of the tree: it is the largest integer h such that x_h appears as an indeterminate in the polynomial. The degree of x_h is the number of children of the root of the tree. We can factor the polynomial with respect to x_h, placing it in the form $(x_h - P_{v_1}) \cdots (x_h - P_{v_k})$; then the P_{v_i} are the polynomials that define the subtrees, so we can recurse on them until we reach the leaves x_0. Since each polynomial has a unique factorization, this gives a unique reconstruction of the tree.

To determine whether T_1 and T_2 are isomorphic, we must check whether the polynomials P_{T_1} and P_{T_2} are equal. We do so by using a Freivalds'-like technique to determine whether $P_{T_1} - P_{T_2} = 0$ everywhere. First, note that the polynomial has total degree n, where n is the number of nodes in both trees, since each variable x_i appears once for each node with height i. So we set $k = n^2$ and and perform all of our further computations modulo k. We select h numbers randomly from the field $\mathbb{Z}_k$ and assign them to the variables $x_1, \ldots, x_h$, and evaluate $P_{T_1} - P_{T_2}$ for this choice of variables. If the result is zero, the trees are likely isomorphic; if not, they are definitely non-isomorphic. By

need a prime.

the Schwartz-Zippel theorem, this has a probability of (one-sided) error if the trees are actually non-isomorphic of $\frac{n}{k} = \frac{n}{n^2} = \frac{1}{n}$, so the algorithm gives the correct result with high probability.

The runtime of this algorithm is then simply the time it takes to evaluate the polynomial for each of the trees modulo k, and check whether they are equal. We do not write down the polynomial in symbolic form, but rather evaluate it for each leaf, then at each vertex of successively greater height. We assume that our machine word is large enough that we can perform multiplications modulo k in constant time. For each leaf, the value is simply the value x_0; for each internal node of height h we subtract the value of the polynomial for each child from the value of x_h then multiply them together. Since this performs one multiplication per each child of some node, it requires $\Theta(n)$ time overall.