CS2020: Data Structures and Algorithms (Accelerated)

Discussion Group Problems for Week 6

For: March 9/10, 2011

Problem 1. (Calculating Probabilities.)

For the following problem, assume that the hash function h satisfies the simple uniform hashing assumption.

Problem 1.a. Assume that hash table T contains $m = 100$ buckets, $n = 50$ of which are full. Assume we are resolving collisions via open addressing, and that the hash function we use satisfies the uniform hashing assumption. What is the exact probability (to 4 decimal places) that an insertion finds an empty bucket within 3 tries?

Problem 1.b. Assume that hash table T is of size $m = n^2$ and has n full buckets. We insert n new elements. Prove the following: the probability that none of the n elements collides with an existing element is $> 1/10$.

Problem 1.c. Assume that hash table T contains n buckets, all initially empty, and we resolve collisions by chaining. Now assume that we add n items to table T. Let B be the expected number of empty buckets (after adding all n elements). Show that: $n/3 < B < n/2$.

Problem 1.d. Assume that hash table T contains n buckets, all initially empty, and we resolve collisions by chaining. Now assume that we add $2n \log n$ items to table T. Show that all n buckets are non-empty with probability at least $(1 - 1/n)$.

Problem 2. (Signatures.)

Consider a hash table T containing $4n$ buckets. As in a Bloom filter, we are not going to store the items themselves in the hash table. Instead, we will only store a signature of each item in the hash table. This should let us get a lower error rate (than if we only stored a binary value), and yet still use less space than storing the entire item in the hash table.

Assume that we have two hash functions: $g : U \to 4n$, which we use to assign elements to buckets, and (for some integer s), $h_s : U \to 2^s$, which we use to generate signatures. Notice that the hash function h_s maps every element to an integer in the range $\{0, \ldots, 2^s - 1\}$. When adding

an element v to our data structure, we write the value $h_s(v)$ to the bucket $g(v)$. When querying the data sturcutre for an element w, we first check the bucket $g(w)$; if it contains a signature x, we then compare $x == h_x(w)$ to determine whether to return true or false.

Problem 2.a. How much space does it take to store the hash table?

Problem 2.b. If there are n items stored in the hash table, and we query whether an element is in the table, what is the probability of a false negative, using this scheme?

Problem 2.c. If there are n items stored in the hash table, and we query whether an element is in the table, what is the probability of a false positive, using this scheme?

Problem 2.d. If there are n items stored in the hash table, and we want to achieve an error rate of at most $p < 1$ (in terms of either false positives or false negatives), then how much space is needed to store the table?

Problem 2.e. (Optional.) Compare the space usage to a Bloom Filter that also achieve error probability $\leq p$, for some constant p. Which data structure uses less space?

Problem 3. Alice is an engineer at Cisco, designing routers. Her routers are going to be used as part of the core internet infrastructure, and will be handling approximately forty million packets per second. Today, there are about 3.5 billion IP addresses in use. Her job is to count how many different IP addresses use her router every day (i.e., how many different computers with unique IP addressses send packets that traverse her router).

At first, she tried the simple solution of maintaining a big list of IP addresses: every time a new packet arrives, she checks whether the IP address is in the list; if not, she adds it to the list. At the end of the day, she simply reports the size of the list.

Problem 3.a. Discuss the (multiple) problems with her first approach.

Problem 3.b. In order to improve performance, Alice decides it is acceptable to report an *approximate* number of IP addresses, rather than an exact number. If the result is off by a factor of 2, that is acceptable. For example, if there are n IP addresses that use the router in a day, her algorithm must report some value n' where $n' \geq n/2$ and $n' \leq 2n$. Explain how Alice might use a Bloom filter to solve the *approximate* counting problem. How long (asymptotically) does it take

to process a new packet? In qualitative terms, how does the approximation ratio (in this case, 2) affect the size of the Bloom filter?

Problem 3.c. (Optional.) Analyze the size of the resulting Bloom filter.