Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
Ankur Moitra

Wednesday, February 10, 2016

Problem Set 1

Due: Wednesday, February 17, 2016 – 7 pm
Outside Stata G5

Collaboration policy: collaboration is *strongly encouraged*. However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator.
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 or 4 people in a given week.
4. Write each problem in a separate sheet and write down your name on top of every sheet.
5. **No bibles. This includes solutions posted to problems in previous years.**

1 Hashing Bashing

Two-level hashing is nice because, for n items, it can achieve perfect hashing with just $O(n)$ buckets. However, it does have the drawback that the perfect hash function has a lengthy description (since you have to describe the second-level hash function for each bucket).

(a) For a universe U of size $|U|$, how many bits are required to describe a perfect two-level hash function that is implemented using 2-universal hash functions? We're just looking for big-O notation.

Consider the following alternative approach to producing a perfect hash function with a small description. Define *bi-bucket hashing*, or *bashing*, as follows. Given n items, allocate *two* arrays of size $O(n^{1.5})$. When inserting an item, map it to one bucket in *each* array, and place it in the emptier of the two buckets.

(b) Suppose a *fully random function* is used to map each item to buckets. Prove that the expected number of collisions is $O(1)$. **Hint:** What is the probability that the k^{th} inserted item collides with some previously inserted item?

(c) Show that bashing can be implemented efficiently, with the same expected outcome, using random hash functions from 2-universal hashing. How large is the description of the resulting function?

(d) Conclude an algorithm that requires just $O(n)$ hash function evaluations in expectation to identify a perfect bash function for a set of n items.

(e) **(Optional)** Generalize the above approach to use less space by exploiting tri-bucket hashing (trashing), quad-bucket hashing (quashing), and so on.

2 Sampling with Few Bits

Consider the problem of using a source of unbiased random bits to generate a sample from the set $S = \{1, \ldots, n\}$ such that element i is chosen with probability p_i.

(a) Suppose $n = 2$ (so $p_2 = 1 - p_1$). Give a scheme that uses $O(1)$ bits in expectation to choose one of the two items. **Hint:** start with an easy scheme that uses a possibly infinite number of random bits to choose the item. Then make it lazy, generating/examining only enough bits to uniquely identify the item that would be selected by the infinite sequence. What is the probability looking at a new bit lets you stop? Analyze the expected number of bits you will actually examine.

(b) Generalize this scheme to sample from n elements using $O(\log n)$ random bits in expectation per sample, regardless of the values of the p_i.

(c) Prove that for certain p_i and n (for example, a uniform sample from $\{1, 2, 3\}$), if you only have unbiased random bits, it is *impossible* to generate an appropriately distributed sample using a finite number of bits *in the worst case*.

3 Lower bound for Balls and Bins

In class we showed that n balls in n random bins see a maximum load of $O(\log n / \log \log n)$. Show this bound is tight.

(a) Show there is a $k = \Omega(\log n / \log \log n)$ such that bin 1 has k balls with probability at least $1/\sqrt{n}$. An inequality that might be helpful: $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k$.

(b) Prove that conditioning on the first bin not having k balls only increases the probability that the second bin does, and so on. Conclude that with high probability, some bin has $\Omega(\log n / \log \log n)$ balls. When we say "high probability", we mean with probability $> (1 - 1/n)$, although your bound could be much higher.

4 Balls and Bins with replacement

Consider the following process for matching n jobs to n processors. In each round, every job picks a processor at random. Each processor can only execute one job per round, so if there

is contention for a processor, one (arbitrary) job gets executed by the processor while the others are refused. The unexecuted jobs *back off* and then try again in the next round. If only a single job happens to match to a processor, that job gets executed. Jobs only take one round of time to execute, so at the beginning of every round all the processors are available.

(a) For $\alpha \leq 1$, show that if αn jobs begin a round, then with probability $> (1 - 1/n^2)$ only $\alpha^2 n$ will not be processed, as long as $\alpha^2 n > c \log n$ for some fixed constant c. **Hint:** consider adding the jobs one at a time, and upper bound the probability that a job is assigned to a previously occupied processor.

(b) Conclude that all the jobs finish executing with high probability after $O(\log \log n)$ steps. Again "high probability" means with probability $> (1 - 1/n)$. Here you will need to handle the case when the number of jobs remaining is very small and thus $\alpha^2 n$ is not greater than $c \log n$.

5 Consistent Hashing with Inconsistent Views

In distributed caching systems, it is hard to keep track of which machines are up or down. Different clients may learn about different machines' states at different times. And if different clients have different opinions about which machines are up, they will have different opinions about which machine to contact to retrieve a given item.

In particular, suppose that there are n clients and m servers currently online. The clients are going to choose which machine to query for a data item based on the consistent hashing scheme discussed in class. However, each client only has probability $1/2$ of knowing about any given server (and all mn of these events, of every client knowing about every server, are independent). Prove that with high probability, $O(\log n)$ machines will be asked to deal with any given data item, regardless of the number of clients interested in that item. As in class, assume that the underlying hash function used to map items to the ring produces independent random values for different keys.