10 | 10 | 5 | 10

6.856

Randomized Algorithms

2005/03/02

Dan Ports

drkp@mit.edu

Collaborators: {sarahl, gpickard}

Problem Set 4

Problem 1:

There are $\binom{m}{n}$ possible subsets of m that need to be hashed perfectly by at least one hash function in a perfect hash family. We will show that any family of the given size can only hash a smaller number.

Consider any arbitrary hash function. For each of the n hash values, the hash function must specify which of the m elements map to this value. The number of sets perfectly hashed by this function is the product of the number of elements mapping to each value. This is maximized when distributed evenly, i.e. when m/n elements map to each value. Then the total number of sets perfectly hashed is $\left(\frac{m}{n}\right)^n$.

Since the size of the hash function is polynomial, there exists some k such that the size of the hash family is $O\left(m^k\right)$. Then the total number of sets perfectly hashed is bounded above by $O\left(m^k\right)\left(\frac{m}{n}\right)^n$ by the union bound.

All subsets are perfectly hashed if the following constraint is met:

$$\begin{aligned} O\left(m^k\right)\left(\frac{m}{n}\right)^n &\geq \binom{m}{n} \\ &\geq \left(\frac{em}{n}\right)^n \\ O\left(m^k\right) &\geq e^n \end{aligned}$$

Suppose this is true. Then there exists some c such that

$$\begin{aligned} cm^k &\geq e^n \\ \log c + k \log m &\geq n \\ \log m &\geq \frac{n - \log c}{k} \end{aligned}$$

Since $m \leq 2^{o(n)}$, $\log m = o\left(n\right)$. Then we have $o\left(n\right) \geq \frac{n - \log c}{k}$, which is a contradiction. So no such perfect hash family exists.

Problem 2:

a) Consider the kth item inserted. Let A_i be the indicator random variable representing the event that item i hashes to the same bucket in the first hash table. Since the hash function is 2-universal onto an array of size $n^{3/2}$, A_i is 1 with probability $\frac{1}{n^{3/2}}$. Note that the number of items in this bucket is bounded above by $\sum_{i=1}^{n} A_i$ (it is an upper bound since not all n items may have been inserted, and some may be in the other hash table.) So the expected size of the bucket is

$$\mathrm{E}\left[\text{bucket size}\right] \leq \mathrm{E}\left[\sum_{i=1}^{n} A_i\right] = \sum_{i=1}^{n} \Pr\left[A_i = 1\right] = \frac{n}{n^{3/2}} = \frac{1}{n^{1/2}}$$

b) The expected bucket size in the first array is $\frac{1}{n^{1/2}}$, so by the Markov bound, the probability that the bucket is non-empty, i.e. has size at least 1, is no greater than $\frac{1}{n^{1/2}}$. The probability that the

bucket in the second array is non-empty is also at most $\frac{1}{n^{1/2}}$. Thus the probability that the buckets in both arrays are non-empty is the product of these probabilities, $\frac{1}{n}$.

c) Suppose we choose the arrays to have size $2n^{3/2}$. This choice of constant factor makes the probability of any particular item's buckets both being non-empty $\frac{1}{2n}$. Since there are n items, by the union bound, the probability that any item's bucket will be non-empty will be at most $\frac{n}{2n} = \frac{1}{2}$. So the probability that no item collides with another is at least $\frac{1}{2}$

d) The above claim shows that for a random pair of 2-universal hash functions, the probability that there are no collisions is at least $\frac{1}{2}$. So the expected number of attempts required before finding a satisfactory pair of 2-universal hash functions is constant.

Once such a pair is found, the hash function can be represented in four machine words: two to describe the 2-universal hash functions for each array.

e) Now suppose there are k arrays of size $2n^{1+1/k}$. By the same reasoning, the expected number of other items in any given item's bucket in the first array is $\frac{n}{2n^{1+1/k}} = \frac{1}{2n^{1/k}}$. Since there are now k buckets, the probability that they will all be non-empty for any given item will be $\left(\frac{1}{2n^{1/k}}\right)^k = \frac{1}{2^k n}$. Applying the union bound over all n items, the probability that there will be no collision is $\frac{1}{2^k}$, so a constant number of hash function set attempts are required.

Problem 3:

Suppose now that instead of maintaining a binary tree of bucket positions, we instead maintain a size m array. We divide the keyspace into m equally-sized intervals, and store all bucket positions in a particular interval in a binary tree pointed to by the corresponding index in the array. Then, to perform a LOOKUP, we find the interval corresponding to the item's hash value, use this to index into the array, and search the corresponding binary tree. If the binary tree is empty, then the associated bucket is not in this interval of the keyspace, and we must search the next interval in the array (and so on).

First we consider the amount of time required to search one binary tree. The number of items in the tree is the number of bucket positions in the associated $1/m$ interval. This is precisely the problem of throwing m balls into m bins — the expected number of items in the tree will be $O(1)$ and $O(\log n)$ with high probability. Since we are using a balanced binary tree, operation costs are logarithmic in the size of the tree, and so performing the search requires $O(1)$ time in expectation and $O(\log \log n)$ time with high probability.

Now we consider the number of empty trees that need to be searched (noting that observing that a tree is empty and moving on to the next requires constant time). The probability that a tree is empty is the probability that no bucket positions lie in the associated interval. This is $\left(1 - \frac{1}{m}\right)^m \approx \frac{1}{e}$. So in expectation at most $\frac{1}{1-\frac{1}{e}}$ intervals must be inspected. The probability that $x+1$ intervals must be inspected is the probability that x adjacent intervals contain no elements, or $\left(\frac{1}{e}\right)^x$. If we choose $x = k \log \log n$, then the probability is $\frac{1}{(\log n)^k}$, which assures us that the search time is $O(\log \log n)$ with high probability. Thus the algorithm runs in constant time in expectation and $O(\log \log n)$ time with high probability.

5/10

NOT whp.

Problem 4:

Fix some specific data item and write the machines in order as $m_1, m_2, m_3, \ldots$, where m_1 is the successor of the data item in question, m_2 is the successor of m_1, etc. Observe that any particular client that knows about half of the machines and is attempting to access the item will contact

machine m_i, where i is the smallest value such that the client knows about m_i.

Consider an arbitrary client. The client knows about half of the machines, so the probability that it knows about m_1 is $1/2$. The probability that it knows about m_k given that it does not know about m_1 through m_{k-1} is $\frac{n/2}{n-k+1}$, which is strictly greater than $1/2$. So the random variable giving the minimum value of i such that the client knows about m_i is stochastically dominated by the geometric distribution with mean 2.

The probability that the specific client contacts a machine beyond the dth one is the probability that it does not know about m_1 through m_{d-1}, which is $(1/2)^{d-1}$. If we choose $d = (1+\epsilon)\lg n + 1$, then the probability of contacting a machine beyond the dth one is $\frac{1}{n^{(1+\epsilon)}}$.

Now we consider the probability that *any* client contacts a machine beyond the dth one. By the union bound, this is

$$\frac{n}{n^{(1+\epsilon)}} = \frac{1}{n^{\epsilon}}$$

Since $d = O(\log n)$, with high probability, $O(\log n)$ machines will receive requests for any particular data item.

use another letter