6.856 — Randomized Algorithms

Handout #11, March 25, 2011 — Homework 4 Solutions

1.

The key difference is that if $\beta_i n$ bins have height h then the probability a ball chooses all height h bins drops to β_i^d. Thus, the "expected number" of height $h+1$ bins is like $\beta_{i+1}n$ where $\beta_{i+1} = \beta_i^d$. This gives $\beta_i = (\frac{1}{4})^{d^i}$ which becomes $O(1/n)$ at $i = O(\log_d \log n)$.

The rest of the proof is unchanged; in order to deal with the conditioning we work with parameters $\beta_{i+1} = (2\beta_i)^d$.

2.

Let $h()$ be an arbitrary hash function mapping elements in $M = x_1, \ldots, x_m$ to elements in $N = y_1, \ldots, y_n$. Let h_i be the number of elements in M mapped to y_i. Thus, the number of subsets of size n in M perfected hashed by $h()$ is $H = \prod_i h_i$. Subject to the constraint that $\sum_i h_i = m$, H achieves the maximum of $\left(\frac{m}{n}\right)^n$ when $h()$ evenly divides the elements or $h_i = \frac{m}{n}$. Since there are $\binom{m}{n}$ unique subsets of size n in M and each hash function can perfectly hash at most $\left(\frac{m}{n}\right)^n$ subsets, we need a hash family of size at least $\binom{m}{n} / \left(\frac{m}{n}\right)^n$ in order to find a perfect hash function for each possible subset. Thus, if $2n \leq m \leq 2^{o(n)}$, the size of a perfect hash family required is bounded by:

$$\begin{aligned}
\frac{\binom{m}{n}}{\left(\frac{m}{n}\right)^n} &= \frac{\frac{m!}{n!(m-n)!}}{\left(\frac{m}{n}\right)^n} \approx \frac{\sqrt{2\pi m}\left(\frac{m}{e}\right)^m}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \sqrt{2\pi(m-n)}\left(\frac{m-n}{e}\right)^{m-n}\left(\frac{m}{n}\right)^n} = \frac{1}{\sqrt{2\pi n}}\left(\frac{m}{m-n}\right)^{m-n+\frac{1}{2}} \\
&> \frac{1}{\sqrt{2\pi n}}\left(1+\frac{n}{m-n}\right)^{m-n} = \frac{1}{\sqrt{2\pi n}}\left[\left(1+\frac{n}{m-n}\right)^{\frac{m-n}{n}}\right]^n \\
&\geq \frac{1}{\sqrt{2\pi n}}\left[e\left(1+\frac{n}{m-n}\right)^{-\frac{1}{2}}\right]^n \geq \frac{1}{\sqrt{2\pi n}}\left[\frac{e}{\sqrt{2}}\right]^n = 2^{\theta(n)}
\end{aligned}$$

We have demonstrated that the required size of a perfect hash family is at least exponential in n. If $m \leq 2^{o(n)}$, any polynomial in m is also $\leq 2^{o(n)}$ since $m^c \leq 2^{c \cdot o(n)} = 2^{o(n)}$ for any constant c. Thus, a hash family of size only polynomial in m is not large enough be be exponential in n. Therefore, there is no perfect hash family mapping from m to n of size polynomial in m.

3.

As hinted, we will use a main table and an overflow table. After k probes of the main table, if we have not found an empty cell, we place the item in the overflow table.

If you have $(1+\varepsilon)n$ space, you can build a main table of size n and a cuckoo-hash table of size εn. By the argument in class, the cuckoo hash table can hold $\varepsilon n/2$ items with constant worst-case lookup time. We're going to guarantee that the "main" table holds only $(1-\varepsilon/2)n$ items by the simple rule that if the main table gets that full, we immediately place other incoming items in the cuckoo hash table.

Assuming the main table has the claimed limit, k probes to it will fail to find an empty bucket with probability $(1-\varepsilon/2)^k$. If we arrange for (say) $(1-\varepsilon/2)^k \leq \varepsilon/4$, then the probability that an item fails to find an empty space is $\varepsilon/4$, so the expected number of items that fail to find a space, and get kicked into overlow, is $\varepsilon n/4$. Thus a chernoff bound tells us that it is at most $\varepsilon n/2$ with (exponentially) high probability, so the cuckoo table will never get too full to operate in constant time (w.h.p.). Solving, we find that $k = \log(\varepsilon/4)/\log(1-\varepsilon/2) = O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$.

4.

To achieve evaluation time of $O(1)$ in expectation and $O(\log\log m)$ with high probability, we will modify the consistent hashing algorithm as follows. First, break the ring into m equal sized intervals. Next, associate with each interval the buckets that overlap the interval. Note that the number of buckets associated with each interval is at most 1 more than the number of bucket boundaries that fall within the interval. To find the bucket associated with a particular item, use the hash function to map the item to a number between $[0,1]$ and find the bucket responsible for the item among the buckets associated with the interval that the item falls in. Thus, the performance of this lookup is dependent on the number of buckets associated with the interval. Specifically, if we pre-order the buckets within each interval, we can find the bucket responsible for a given item using binary search in $O(\log b)$ time where b is the number of buckets in the interval.

Since the bucket boundary positions are randomly selected, the problem of finding the expected and maximum number of bucket boundary positions within each interval reduces to the m balls in m bins problem. Thus, we can conclude that the expected number of boundary positions in each interval is $O(1)$ and the maximum number of boundary positions in any interval is $O\left(\frac{\log m}{\log\log m}\right)$ with high probability. With at most $1+1=2$ buckets in each interval in expectation, it takes 1 comparison, or $O(1)$ time, to determine which of the 2 buckets an item maps to. We maintain pointers in each empty interval to the next non-empty interval to allow fast($O(1)$) search. These pointers can be maintained when inserting/deleting machines. With at most $O\left(\frac{\log m}{\log\log m}\right)+1$ buckets in each interval with high probability, it takes $\log\left[O\left(\frac{\log m}{\log\log m}\right)+1\right]=O(\log\log m)$ comparisons to determine the bucket an item maps to with high probability. Thus, a consistent hash function can be evaluated in $O(1)$ time in expectation and $O(\log\log m)$ time with high probability.

5.

Let there be m machines and n clients interested in a specific data item. Using the consistent hashing scheme, each machine and data item is mapped to the cyclical interval $[0,1]$. For convenience, label the machines $1,\ldots,m$ in order of appearance after the position of the data item. Thus, machine 1 owns the data item. However, if a particular client believes that machines $1,\ldots,k$ are down, it will query machine $k+1$ for the data.

Let $z_i^j=1$ if client j believes machine i is up and 0 otherwise. Thus, for $k<\frac{m}{2}$, the probability that a client j queries machine $k+1$ is bounded by:

$$\begin{aligned}
\Pr[[]\text{client } j \text{ queries machine} \geq k+1] &= \Pr[[]z_1^j=z_2^j=\cdots=z_k^j=0] \\
&= \frac{\#\text{ ways to choose machines such that } z_1^j=\cdots=z_k^j=0}{\#\text{ ways to choose } \frac{m}{2} \text{ down machines}} \\
&= \frac{\binom{m-k}{\frac{m}{2}-k}}{\binom{m}{\frac{m}{2}}} = \frac{\frac{(m-k)!}{(\frac{m}{2}-k)!(\frac{m}{2})!}}{\frac{m!}{(\frac{m}{2})!(\frac{m}{2})!}} = \frac{\left(\frac{m}{2}\right)\left(\frac{m}{2}-1\right)\cdots\left(\frac{m}{2}-k+1\right)}{(m)(m-1)\cdots(m-k+1)} \\
&= \left(\frac{1}{2}\right)^k \frac{(m)(m-2)\cdots(m-2k+2)}{(m)(m-1)\cdots(m-k+1)} \\
&< 2^{-k}
\end{aligned}$$

Choosing $k=c\log n$ and applying the union bound, the probability that any of the n clients query machine $\geq k+1$ is bounded by $n2^{-k}=n^{-(c-1)}$. Thus, with high probability, no client will query machine $\geq O(\log n)$. Alternatively, at most $O(\log n)$ machines will be queried with high probability.