6.856 — Randomized Algorithms

David Karger

Handout #5, Mar. 2, 2011 — Homework 4, Due 3/9

1. Improving the two-choice bound.

   (a) In class we proved that the two-choices approach improves the maximum load to $O(\log \log n)$. A generalization is that choosing the least loaded of d choices reduces the maximum load to $O(\log_d \log n)$. Explain what changes to the proof are needed to derive this result. Give only the diffs; do not bother writing a complete proof.

   (b) **Optional.** Suppose that instead of making two choices at random, you divide the bins into a left and right half and break all ties by putting items in the left bin. Show that the maximum load improves by a constant factor, from $O(\log \log n / \log 2)$ to $O(\log \log n / 2 \log \phi)$ where $\phi = (1 + \sqrt{5})/2$ is the golden ratio. **Hint:** for the number of height i bins on each side, use different recurrences β_i for the left side and γ_i for the right side. Show that $\beta_{i+1} \leq c_1 \beta_i \gamma_i / n^2$ while $\gamma_{i+1} \leq c_2 \beta_{i+1} \gamma_i / n^2$.

   (c) **Optional.** Generalize to d bins, showing a load of $O((\log \log n)/d)$.

2. Show that there is no perfect hash family mapping from m to n of size polynomial in m (i.e., that can be represented in a constant number of $\log m$-bit machine words) if $2n \leq m \leq 2^{o(n)}$. **Hint:** use a probabilistic existence proof to show that for any family of that size, there exists a set that does not get perfectly hashed.

3. Cuckoo hashing is nice, but does cost a factor of two in space. Develop a related approach that uses less space while still guaranteeing worst-case constant-time lookups. Use the following ideas:

   - Probing more than twice in a table increases the chances of finding an empty cell.
   - If after some probes you fail to find an empty cell, move the failed item into an "overflow" table that uses cukoo hashing

   Use this to achieve constant-time lookups using only $(1 + \epsilon)$ space for any constant ϵ. Determine the best tradeoffs you can between number of probes required and amount of space used.

4. **This problem should be solved without collaboration.** In class we argued that a consistent hash function could be evaluated in $O(\log m)$ time by putting the bucket IDs in a binary search tree. Argue that this can be improved to $O(1)$ time in expectation, and $O(\log \log m)$ with high probability. **Hint:** Use the fact that the bucket positions are random, and consider breaking the ring into m equal sized intervals that could be represented as a size-m array.

5. Another problem with the distributed caching system we discussed is that 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. Suppose that every client knows about half of the machines that are up at a given time. 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.