

Problem Set 4 Solutions

Problem 1. (a) 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 $\beta_{i+1} = (2\beta_i)^d$.

(b) Following the hint, define $\beta_4 = \gamma_4 = 1/4$. Then let $\beta_{i+1} = \beta_i \gamma_i$ and $\gamma_{i+1} = \beta_{i+1} \gamma_i$. By induction, the number of left-side bins of height i is at most $2\beta_i n$ and the number of right-side bins of height i is at most $2\gamma_i n$. The base case is clear, and the the induction step follows the same ideas we used in class for the original two choice bound.

Now, we claim $\beta_{4+i} = (1/4)^{F_{2i}}$ and $\gamma_i = (1/4)^{F_{2i+1}}$, where F_i are the Fibonacci numbers with $F_0 = F_1 = 1$. The base case for $i = 1$ follows directly from applying one step of the recursion. For the inductive step, we have:

$$\beta_{i+1} = \beta_i \gamma_i = (1/4)^{F_{2i}} (1/4)^{F_{2i+1}} = (1/4)^{F_{2i+2}} = (1/4)^{F_{2(i+1)}}$$

$$\gamma_{i+1} = \beta_{i+1} \gamma_i = (1/4)^{F_{2i+2}} (1/4)^{F_{2i+1}} = (1/4)^{F_{2(i+1)+1}}$$

as desired. It is well known that $F_{2i} \approx \phi^{2i}/\sqrt{5}$, and so we have that

$$\beta_{(\log_\phi \log n)/2} = (1/4)^{F_{\log_\phi \log n}} \approx 2^{-2\phi^{\log_\phi \log n}/\sqrt{5}} = n^{-2/\sqrt{5}}$$

Thus, at most $n^{1-2/\sqrt{5}}$ balls on the left side have height at least $(\log_\phi \log n)/2$ and we can use the union bound to show that with high probability, no ball will have height greater than $(\log_\phi \log n)/2 = \log \log n / (2 \log \phi)$.

- Problem 2.** (a) For any fixed i, j, k where $i < j < k$, the probability that balls i, j, k are placed uniformly at random all in the same bin is $1/n^2$. The number of such triples is $\binom{n}{3}$, so $E[\sum C_{i,j,k}] = \binom{n}{3}/n^2$.
- (b) Let S be the set of balls in the bin containing m items. There are $\binom{m}{3}$ ways to choose $i < j < k$ from S . These triples of balls are in the same bin (in other words, $C_{i,j,k} = 1$ for these triples i, j, k), so $\sum C_{i,j,k} \geq \binom{m}{3}$.
- (c) If a bin has $cn^{1/3}$ balls, then by part (b),

$$\sum C_{i,j,k} \geq \binom{cn^{1/3}}{3} \geq ((c/2)n^{1/3})^3/6 = \frac{c^3}{48}n.$$

By Markov's Inequality,

$$\begin{aligned} \Pr(\text{some bin has size at least } cn^{1/3}) &\leq \Pr\left(\sum C_{i,j,k} \geq \frac{c^3}{48}n\right) \\ &\leq E[\sum C_{i,j,k}] / \left(\frac{c^3}{48}n\right) \\ &= \frac{\binom{n}{3}}{n^2} / \left(\frac{c^3}{48}n\right) \\ &\leq \frac{48}{c^3}. \end{aligned} \tag{1}$$

Therefore, if we pick a large enough constant c , the probability that some bin has at least $cn^{1/3}$ balls will be small (say smaller than $1/3$).

- (d) Let b_i be the bin that ball i falls into. We note from part a that the only assumption on the randomness needed is that

$$\Pr(i, j, k \text{ all in the same bin}) = nP[b_i = b_j = b_k = b] = \tag{2}$$

$$= n \times P[b_i = b]P[b_j = b]P[b_k = b] = n^{-2} \tag{3}$$

which holds so long as the hash function is 3-way independent.

- (d) Let $z_1 = h(x_1), z_2 = h(x_2), z_3 = h(x_3)$ be the hash values of $x_1 \neq x_2 \neq x_3$. It follows by definition that they satisfy the linear system

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} \begin{bmatrix} c \\ b \\ a \end{bmatrix} \equiv V \begin{bmatrix} c \\ b \\ a \end{bmatrix} \tag{4}$$

if V , a 3×3 Vandermonde matrix, is non-singular, then it can be inverted to obtain

$$\begin{bmatrix} c \\ b \\ a \end{bmatrix} = V^{-1} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \tag{5}$$

in this manner, there is a one-to-one mapping between each of the p^3 assignments of z_1, z_2, z_3 and the p^3 assignments of a, b, c . Since a, b, c are uniformly random in \mathbb{Z}_p , then the triples of hash values z_1, z_2, z_3 are uniformly random, and thereby mutually independent. To conclude this proof it suffices to show that the Vandermonde matrix with parameters $x_1 \neq x_2 \neq x_3$ is invertible, i.e. its determinant is non-zero. Note

$$\det V = \prod_{1 \leq i < j \leq 3} (x_j - x_i) \text{ and } (x_j - x_i) \neq 0 \Rightarrow \det V \neq 0 \quad (6)$$

Alternative solution We can follow the approach for 2-universal hash functions shown in class to and derive an analogous statement for this 3-universal hash function. Let us begin by noting that since

$$h(x) = ax^2 + bx + c \quad (7)$$

and a, b, c is uniformly random, then $h(x)$ is uniformly random. Now, let us re-write the joint probability of 3 hashes, conditioning on the result of h_2 and h_3

$$\mathbb{P}_{a,b,c} \left[h(x_1) = z_1, h(x_2) = z_2, h(x_3) = z_3 \right] = \quad (8)$$

$$= \mathbb{P}_{a,b,c} \left[h(x_1) = z_1, h(x_2) = z_2 \mid h(x_3) = z_3 \right] \times \mathbb{P}_{a,b,c} \left[h(x_3) = z_3 \right] = \quad (9)$$

$$= \mathbb{P}_{a,b,c} \left[h(x_1) = z_1 \mid h(x_2) = z_2, h(x_3) = z_3 \right] \mathbb{P}_{a,b,c} \left[h(x_2) = z_2 \mid h(x_3) = z_3 \right] \mathbb{P}_{a,b,c} \left[h(x_3) = z_3 \right] \quad (10)$$

Note that conditioning on $h(x_3) = ax_3^2 + bx_3 + c = z_3$, implies $h(x_2) = z_3 + a(x_2^2 - x_3^2) + b(x_2 - x_3)$. Since $x_2 \neq x_3$, and a, b are uniformly random, then $h(x_2)$ is uniformly random. It follows that the function is 2-universal and

$$\mathbb{P}_{a,b,c} \left[h(x_2) = z_2 \mid h(x_3) = z_3 \right] = \mathbb{P}_{a,b,c} \left[h(x_2) = z_2 \right] \quad (11)$$

Finally, let us consider $h(x_1)$, conditioned on the result of $h(x_2) = z_2$ and $h(x_3) = z_3$. Note that we can re-write their definitions in terms of c to obtain

$$c = z_3 - ax_3^2 - bx_3 = z_2 - ax_2^2 - bx_2 \quad (12)$$

$$\Rightarrow b = \frac{z_3 - z_2}{x_3 - x_2} - a(x_3 + x_2), \text{ and } c = \frac{z_2x_3 - z_3x_2}{x_3 - x_2} + ax_3x_2 \quad (13)$$

and thus we can re-write $h(x_1)$ conditioned on z_2, z_3 as

$$h(x_1) = a(x_3 - x_1)(x_2 - x_1) + \frac{x_1(z_3 - z_2) + z_2x_3 - z_3x_2}{x_3 - x_2} \quad (14)$$

assuming $x_3 \neq x_2 \neq x_1$, and given a uniform at random, then so is $h(x_1)$. It follows

$$\mathbb{P}_{a,b,c} \left[h(x_1) = z_1 | h(x_2) = z_2, h(x_3) = z_3 \right] = \mathbb{P}_{a,b,c} \left[h(x_1) = z_1 \right] \quad (15)$$

and thereby the function is 3-way independent.

Problem 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 $(1 + \epsilon)m$ space, you can build a main table of size n and a cuckoo-hash table of size ϵm . By the argument in class, the cuckoo hash table can hold $\epsilon m/2$ items with constant worst-case lookup time. We're going to guarantee that the "main" table holds only $(1 - \epsilon/2)m$ 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 $(1 - \epsilon/2)^k$. If we arrange for (say) $(1 - \epsilon/2)^k \leq \epsilon/4$, then the probability that an item fails to find an empty space is $\epsilon/4$, so the expected number of items that fail to find a space, and get kicked into overflow, is $\epsilon m/4$. Thus a chernoff bound tells us that it is at most $\epsilon m/2$ with (exponentially) high probability¹, so the cuckoo table will never get too full to operate in constant time (w.h.p.). Solving, we find that $k = \log(\epsilon/4)/\log(1 - \epsilon/2) = O(\frac{1}{\epsilon} \log(\frac{1}{\epsilon}))$.

¹note that the events that each item overflows are dependent, but we can apply the same trick from class to stochastically dominate with i.i.d. random variables

Problem 4. 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 $1, \dots, m$ 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 $1, \dots, k$ 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:

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

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.

Problem 5. 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.

Problem 6. (a) Consider the event of a bashing collision between the k^{th} inserted item and the j^{th} inserted item, for $j < k$. For this event to happen, one of the two candidate locations of item k has to be the same as the location of j (this has probability at most $2/n^{1.5}$), and the other candidate location for item k must contain at least one element (probability $k/n^{1.5}$). Thus, the probability k collides with j is at most $2k/n^3$. By linearity of expectation we get

$$E[\#\text{collisions}] = \sum_{j < k} \Pr(\text{collision between } j, k) \leq$$

$$\sum_{k=1}^n \sum_{j=1}^k \frac{2k}{n^3} \leq \sum_{k=1}^n \frac{2k^2}{n^3} = \frac{2n(n+1)(2n+1)}{6n^3} \leq \frac{5}{6}.$$

Note that a “good bound” in the expected number of collisions is a constant smaller than 1, since by Markov’s inequality it allows us to argue that with constant probability we get 0 collisions and thus a perfect hash.

(b) By inspecting the argument in (a), we see that we have only used the probability for pairwise collisions, which remains the same if instead of a truly random hash function we use one sampled from a pairwise independent family.

In more detail, denote by h_1, h_2 the two hash functions mapping elements into the two tables, and denote by x_i the i^{th} inserted element, for $i = 1, \dots, n$. In (a) we argued that for x_j to collide with x_k , the following two events must occur:

- x_j and x_k collide in at least one of the two hash functions, and the probability for this can be bounded by $\Pr(\text{ } h_1(x_k) = h_1(x_j)) + \Pr(\text{ } h_2(x_k) = h_2(x_j))$.
- In the other hash function (say w.l.o.g. h_1), x_k collides with at least one previously inserted item, and the probability for this can be bounded by $\sum_{i=1}^{k-1} \Pr(\text{ } h_1(x_k) = h_1(x_i))$.

Then, we used the fact that $\Pr(\text{ } h_1(x_k) = h_1(x_j)) \leq 1/n^{1.5}$ for every i (and similarly for h_2), which holds for pairwise independent functions as well as for truly random function. Hence the bound on the expected number of collisions from (a) holds here as well.

To get an efficient construction, we use the pairwise independent hash family presented in class, whose description size is $O(\log m)$ where m is the universe size (as opposed to $O(n \log m)$ for a truly random function).

(c) We sample h_1, h_2 independently from the pairwise independent hash family presented in class, and start bashing the elements into the tables one by one. Meaning, for each item we compute both of the candidate location. If at least one of them is empty, we insert the item there. Otherwise, if both are full (i.e. bashing collision), we abort and restart the algorithm. Each such attempt takes $O(n)$ time, since processing an element takes $O(1)$ time.

By part (b) the expected number of collisions in each attempt is $5/6$, so by Markov's inequality, with probability at least $1/6$ we get less than 1 collision, which means a perfect bash. Hence in expectation we perform at most 6 attempts until a successful one, so the total expected running time is $O(n)$.

Since a pairwise independent hash function (from the family presented in class) has description size $2 \log m$, and we use two such functions, the total description size of the bash is $O(\log m)$.

- (d) If we map our n items to k candidate locations in an array of size $n^{1+1/k}$, our collision odds work out as above and we get a constant number of collisions. Similarly, k random 2-universal hash families, each mapping to a set of size $n^{1+1/k}$, has a constant probability of being perfect for any particular set of items, so the set of all such functions provides a perfect family (of polynomial size for any constant k). This gives a tradeoff of k probes for perfect hashing in space $O(n^{1+1/k})$.

Note that while we can achieve perfect hashing to $O(n)$ space, the resulting family does *not* have polynomial size (since a different, subsidiary hash function must be chosen for each sub-hash-table).