Homework 5

Aleck Zhao

October 12, 2017

1 Hashing (50 points)

We saw in class that universal hashing lets us give guarantees that hold for arbitrary (i.e. worst case) sets S, in expectation over our random choice of hash function. Let's work out some more of those guarantees.

(a) (25 points) Let H be a universal hash family from U to a table of size m. Let $S \subseteq U$ be a set of m elements which we want to hash (so we're hashing the same number of elements as the size of the table). Prove that if we choose h from H uniformly at random, the expected number of pairs $x, y \in S$ that collide is at most $\frac{m-1}{2}$.

Proof. Since H is a universal hash family, we have

$$\mathbf{Pr}[h(x) = h(y)] \leq \frac{1}{m}$$

Let I_{xy} be an indicator equal to 1 if x and y collide for $x \neq y$. Then since the sum of all of these indicators double counts each pair, the expected number of collisions is

$$\begin{aligned}\mathbf{E}\left[\frac{1}{2}\sum_{x\in S}\sum_{y\neq x} I_{xy}\right] &= \frac{1}{2}\sum_{x\in S}\sum_{y\neq x}\mathbf{E}[I_{xy}] = \frac{1}{2}\sum_{x\in S}\sum_{y\neq x}\mathbf{Pr}[h(x)=h(y)] \\ &\leq \frac{m(m-1)}{2}\cdot\frac{1}{m} = \frac{m-1}{2}\end{aligned}$$

as desired. □

(b) (25 points) Prove that with probability at least 3/4, no bin in the table gets more than $2\sqrt{m}+1$ elements.

Hint: use part (a), and consider using Markov's Inequality. To remind you: if X is a nonnegative random variable with expectation $\mathbf{E}[X]$, then $\mathbf{Pr}[X > k\mathbf{E}[X]] < 1/k$ for any $k > 0$. For example, the probability that X is more than 100 times its expectation is less than 1/100.

Proof. Let X be the number of collisions. Suppose some bin had at least $2\sqrt{m}+2$ elements. Then the number of collisions in this bin is at least $\frac{(2\sqrt{m}+1)(2\sqrt{m}+1)}{2} = 2m + 3\sqrt{m} + 1$. The event that some bin has at least $2\sqrt{m}+2$ elements is a subset of the event that $X > (2m + 3\sqrt{m} + 1)$, since it is possible there are more than $2m + 3\sqrt{m} + 1$ collisions with no bin having more than $2\sqrt{m}+1$ elements. Thus,

$$\begin{aligned}\mathbf{Pr}\big[\text{some bin with at least } 2\sqrt{m}+2 \text{ elements}\big] &\leq \mathbf{Pr}\big[X > 2m + 3\sqrt{m} + 1\big] \\ &= \mathbf{Pr}\left[X > \frac{2m+3\sqrt{m}+1}{\frac{m-1}{2}}\cdot\frac{m-1}{2}\right] \\ &\leq \mathbf{Pr}\left[X > \frac{2m+3\sqrt{m}+1}{\frac{m-1}{2}}\cdot\mathbf{E}[X]\right] \\ &< \frac{\frac{m-1}{2}}{2m+3\sqrt{m}+1} \leq \frac{1}{4}\end{aligned}$$

The probability that no bin has more than $2\sqrt{m}+1$ elements is the complement of this, which is at least 3/4, as desired. □

2 Union-Find (50 points)

In class we proved that if we use trees to represent disjoint sets and use both union-by-rank and path compression, then the amortized cost of any operation (Make-Set, Find, or Union) is only $O(\log^* n)$. In this question we'll analyze what happens if we change our data structure in two ways: we do not use path compression, and we do union-by-cardinality rather than union-by-rank.

Slightly more formally, we change our data structure as follows. As before, every node has three values: an element, a parent pointer, and a value. But the value isn't a rank, but is rather the number of nodes in the subtree rooted at it, i.e., the *cardinality* of the subtree. So the root of any tree stores the cardinality of the set represented by the tree. The basic operations work as follows:

- Make-Set(x) simply returns a single node with the element x, where the parent pointer points to itself and the initial cardinality is 1.
- Find(x) follows the parent pointer from x recursively until it ends up at the root (which it knows since only the root will have its parent pointer point to itself), and then it returns the element at this root.
- Union(x, y) first does Find(x) to find the root r_x of the tree containing x and Find(y) to find the root r_y of the tree containing y. If $(r_x \to \textit{cardinality}) \geq (r_y \to \textit{cardinality})$, then we set the parent of r_y to be r_x and set $(r_x \to \textit{cardinality}) = (r_x \to \textit{cardinality}) + (r_y \to \textit{cardinality})$. Similarly, if $(r_y \to \textit{cardinality}) > (r_x \to \textit{cardinality})$, then we set the parent of r_x to be r_y and set $(r_y \to \textit{cardinality}) = (r_x \to \textit{cardinality}) + (r_y \to \textit{cardinality})$.

(a) (25 points) Prove that the worst-case running time of any operation is $O(\log n)$, where n is the number of Make-Set operations (i.e., the number of elements). Hint: can you bound the depth/height of a tree by its cardinality?

Proof. The time of a Find operation is $O(h)$ where h is the height of the tree we are searching in. We claim that the height of a tree is at most the log of its cardinality C, where a singleton element has height 0. Proceed by induction: if $C = 1$, the tree is a singleton, and its height is $\log 1 = 0$. If we have two trees with cardinalities C_1 and C_2, their worst case heights are $\log C_1$ and $\log C_2$, respectively. If $\log C_1 = \log C_2$, then the height of the resulting tree is $\log C_1 + 1 = \log 2C_1 \leq \log(C_1 + C_2)$. Otherwise, if $\log C_1 > \log C_2$, then the resulting height is just $\log C_1 \leq \log(C_1 + C_2)$ and similarly if $\log C_2 > \log C_1$.

Each Make-Set operation takes $O(1)$, and each Find operation takes $O(h) = O(\log n)$, and since each Union operation is just two Finds and a constant, the worst-case running time of any operation is $O(\log n)$, as desired. □

(b) (25 points) We now want to provide a matching lower bound not just on the worst-case running time, but even on the amortized running time. So suppose that we first do n Make-Set operations (so there are n elements). Give a sequence of $O(n)$ Union operations which cumulatively take $\Omega(n \log n)$ time (implying that the amortized cost of an operation is $\Omega(\log n)$). Each union operation should be a union of two distinct sets, i.e., you should never call Union(x,y) on elements x and y that are in the same set.

Solution. First make $n/2$ calls to Union to create $n/2$ sets of size 2. These union calls will always choose 2 singleton sets. These calls each take at least 1 time. Next, call Union $n/4$ times to create $n/4$ sets of size 2, always choosing 2 distinct sets. These calls each take 2 time. Continue this process, where the total number of Union calls is

$$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 1 = n - 1 = O(n)$$

At level i, the time to perform the Unions is at least $\frac{n}{2^i} \cdot 2^{i-1} = n/2$, and there are $\log n$ levels, so the total amount of time is $\frac{n}{2} \log n = \Omega(n \log n)$, and thus the amortized cost per operation is $\Omega(\log n)$. □