princeton univ. F'25 cos 521: Advanced Algorithm Design

Lecture 4: Hashing II

Lecturer: *Huacheng Yu*

Today, we will look at hashing schemes for different settings or which achieve better bounds. We will assume access to fully random hash functions.

1 Consistent Hashing

In this setting, we want to hash m items (which we denote by M) to n buckets (which we denote by N), where $m \gg n \gg 1$. We want to be able to handle insertion and deletion of items *and* buckets. The goal is to design a hash function which

- Hashes $\approx m/n$ items to each bucket,
- Can perform insertion/deletion while relocating few items, and
- Can perform fast lookup.

Consider the non-standard fully random hash function $h : (M \cup N) \to [0, 1)$, where we interpret $[0, 1)$ as a circle (note that h can be discretized to a large range like $[2^{64}]$ with minimal changes to the analysis). For any set of buckets $Y \neq \emptyset$, Define the Y-*successor* of an item x to be the bucket $y \in Y$ which is closest to x in the clockwise direction. Formally,

$$Y\text{-successor of } x \quad = \quad \begin{cases} \arg\min_{y \in Y, h(y) \geq h(x)} h(y) & \{y \in Y : h(y) \geq h(x)\} \neq \emptyset \\ \arg\min_{y \in Y} h(y) & o.w. \end{cases}$$

Then we implement the hash table operations as follows:

- Lookup: given item x and currently inserted buckets Y, compute the Y-successor of x, which can be done with a BST in $O(\log(n))$ time.
- Bucket insertion/deletion: insert or delete the bucket y with key $h(y)$ from the BST in $O(\log(n))$ time.
- Item insertion/deletion: no action needed (can use other data structures if you want to track, e.g., items associated with each bucket).

Let $Z_{y,X,Y}$ be the number of items in bucket y when items X and buckets Y are inserted. Then since h is uniformly random,

$$\mathbb{E}[Z_{y,X,Y}] \quad = \quad \frac{|X|}{|Y|} \ .$$

We will not perform the computation, but we can show that

$$\mathrm{Var}[Z_{y,X,Y}] \quad = \quad O\left(\frac{|X|^2}{|Y|^2}\right) \ .$$

This says the standard deviation of the number of items in each bucket is on the order of the expectation, which is pretty bad.

1.1 Reducing the Variance

The majority of the variance is coming from the fact that there is high variance in the size of the interval between buckets. This can be reduced in the following way: instead of using $h : (M \cup N) \to [0, 1)$, use a hash function over the larger bucket domain of $h : (M \cup (N \times [k])) \to [0, 1)$. Then when we insert/delete a bucket y, we add/remove all k points $h(y, 1), \dots, h(y, k)$ to the circle. We redefine the Y-successor of x to be the "sub-bucket" (y, ℓ) such that $h(y, \ell)$ is closest to $h(x)$ in the clockwise direction, and lookup an item x by finding the bucket y associated with the Y-successor of x.

Let $Z_{y,\ell,X,Y}$ be the number of items in sub-bucket (y, ℓ). Since the effect of turning each bucket y into k sub-buckets $(y, 1), \dots, (y, k)$ is to essentially blow up the size of Y by a factor of k, we have

$$\mathbb{E}[Z_{y,X,Y}] \quad = \quad \sum_{\ell=1}^{k} \mathbb{E}[Z_{y,\ell,X,Y}] \quad = \quad \sum_{\ell=1}^{k} \frac{|X|}{k|Y|} \quad = \quad \frac{|X|}{|Y|}$$

and (since variance of a sum of independent r.v.s is the sum of the variances)

$$\mathrm{Var}[Z_{y,X,Y}] \quad = \quad \sum_{\ell=1}^{k} \mathrm{Var}[Z_{y,\ell,X,Y}] \quad \approx \quad \sum_{\ell=1}^{k} O\left(\frac{|X|^2}{k^2|Y|^2}\right) \quad = \quad O\left(\frac{|X|^2}{k|Y|^2}\right) .$$

Note that the actual variance is

$$\mathrm{Var}[Z_{y,X,Y}] \quad = \quad O\left(\frac{|X|^2}{k|Y|^2} + \frac{|X|}{|Y|}\right)$$

because the number of items hashing to each interval also influences the variance, and this term dominates when the number of sub-buckets becomes large.

Observe that bucket insertions/deletions now take $O(k \log(n))$ time, but as long as $k = \mathrm{poly}(n)$, accesses still only take $O(\log(nk)) = O(\log(n))$ time.

2 Power of Two Choices

Let us return to a simpler setting from last week: we want to hash n items (insertion only) to n (fixed) buckets.

Let $h_1, h_2 : [n] \to [n]$ be independent fully random hash functions from items to buckets, and consider the following hashing scheme: when inserting item x, check which bucket of $h_1(x), h_2(x)$ has fewer items and store it in that bucket.

Last week, we saw that if we just use one hash function, we could only guarantee that the max-load was $O(\log(n)/\log\log(n))$ w.h.p. using a Chernoff bound. We will now show that using the above two hash function scheme, the max-load is only $O(\log\log(n))$ w.h.p., an exponential improvement. Consider the following informal argument:

1. Define β_k to be the number of buckets with $\geq k$ items.

2. Observe that if inserting item x creates a new bucket with $\geq k+1$ items, then buckets $h_1(x), h_2(x)$ must both have $\geq k$ items.

3. The probability that this occurs is $\approx (\beta_k/n)^2$ (conditioned on β_k), so summing over the n items, $\mathbb{E}[\beta_{k+1} \mid \beta_k] \approx \beta_k^2/n$. Note that this is imprecise because after conditioning on β_k, the hash functions h_1, h_2 are no longer fully random.

4. Using Chernoff bounds, we can show that $\beta_{k+1} \lesssim \beta_k^2/n$ w.h.p.

5. Since there are only n items, $\beta_2 \le n/2$. We can then inductively reason that $\beta_k \lesssim n \cdot 2^{-2^{k-2}}$, so $\beta_{O(\log\log(n))} = 0$ w.h.p.

To formalize this reasoning, we first need to define $\beta_k^{(t)}$ to be the number of buckets with $\ge k$ items after the t^{th} item is inserted (to formalize step (2)). Then reasoning as outlined above, we can show that $\beta_k^{(t)} \le \max\{2^{-2^{k-2}}, O(\log(n))\}$ w.p. $1 - \text{poly}(n)$, where the $O(\log(n))$ is necessary for the statement to hold w.h.p. via Chernoff bound. Therefore, for some $k^* = O(\log\log(n))$, we have $\beta_{k*}^{(t)} = O(\log(n))$. We can then directly reason that $\beta_{k^*+1}^{(t)} = 0$ w.p. $1 - \tilde{O}(1/n)$.

What happens if we have d hash functions and store each item in the least occupied bucket among d choices? Then we would have the relation $\beta_{k+1} \lesssim \beta_k^d/n^{d-1}$, which yields $\beta_k \le n \cdot 2^{-d^{k-O(1)}}$, which gives us a max-load of $O(\log\log(n)/\log(d))$. So when $d = O(1)$, we don't see any asymptotic improvement beyond two choices.

2.1 Cuckoo Hashing

Now, suppose we want to store m items into a hash table of size $n \ge 10m$, where each entry of the hash table can only store one item. Let $h_1, h_2 : [m] \to [n]$ be independent fully random hash functions.

Cuckoo hashing is the following scheme: when item x is inserted, store it in $h_1(x)$ or $h_2(x)$ if either is empty. Otherwise, store x in $h_1(x)$, and kick out the old item in $h_1(x)$ to its alternate slot. Repeat until we find an empty entry for the kicked out item(s) or a cycle. If we find a cycle, sample new h_1, h_2 and rebuild the entire hash table.

To analyze cuckoo hashing, we can treat the n entries of the hash table as vertices and the m items as edges of a graph, where item x forms an edge between vertices $h_1(x), h_2(x)$.

Theorem 1. *The probability that we find a cycle during insertion is at most* $1/4$.

Proof. Observe that a cycle is found during insertion only if a cycle exists in the graph induced by the hash table. Since h_1, h_2 are independent fully random hash functions, each edge of the graph exists w.p. at most $m/\binom{n}{2} \le (n/10)(2/n^2) = 1/(5n)$.

Therefore, for a fixed cycle of length k, the probability it exists is at most $1/(5n)^k$ (all k of its edges need to be present, presence of edges is negatively correlated). As there are at most n^k cycles of length k, the probability that any cycle of length k exists is at most 5^{-k} by a union bound. Another union bound shows that the probability that a cycle of any length exists is at most $\sum_{k\in[n]} 5^{-k} \le 1/4$. $\square$

In a similar vein, the time to insert an item x is bounded by the size of the connected component the edge $h_1(x), h_2(x)$ belongs to. Since each edge of the graph exists w.p. at most $1/(5n)$, the graph is very sparse, and it turns out that the expected size of each connected component (and hence the expected insertion time) is $O(1)$.