Recitation 4: Hashing

1 Universal Hashing

Suppose that we are hashing n items that belong to some universe U (where $|U|$ is much larger than n) into the range $\{1, \ldots, m\}$. A family of hash functions $\mathcal{H}$ is called a *universal hash function family* iff

$$\Pr_{h \in_R \mathcal{H}}[h(k) = h(k')] \leq \frac{1}{m} \text{ for all } k \neq k'$$

In other words, $\mathcal{H}$ is a universal hash function family iff the probability that the keys k and k' collide when we pick a random hash function $h \in_R \mathcal{H}$ is small—as small as we would expect if we just hashed k and k' to random elements of $\{1, \ldots, m\}$.

In lecture, we saw that when h is sampled from a universal hash function family we have very few collisions—about $\binom{n}{2} \cdot \frac{1}{m}$ total in expectation. Specifically, let us define $C_{k,k'}$ to be the indicator random variable for the event that keys k and k' collide, i.e.

$$C_{k,k'} = \begin{cases} 1 & \text{if } h(k) = h(k') \\ 0 & \text{otherwise} \end{cases}$$

Note that by the definition of a universal hash function, for any k, k' with $k \neq k'$ we have

$$\mathop{\mathbb{E}}_{h \in \mathcal{H}}[C_{k,k'}] = \Pr_{h \in \mathcal{H}}[h(k) = h(k')] \leq \frac{1}{m}$$

Let $C = \sum_{k=1}^{n-1} \sum_{k'=k+1}^{n} C_{k,k'}$ be the total number of collisions. By linearity of expectation and the universal hashing assumption, we have that

$$\mathop{\mathbb{E}}_{h \in \mathcal{H}}[C] = \sum_{k=1}^{n-1} \sum_{k'=k+1}^{n} \mathop{\mathbb{E}}_{h \in \mathcal{H}}[C_{k,k'}] \leq \binom{n}{2} \cdot \frac{1}{m}$$

1.1 Examples

- Consider a family of hash functions $\mathcal{H} = \{h_1, h_2\}$ that maps a universe $\{a, b, c\}$ to $\{0, 1\}$, where $h_1(a) = h_1(b) = h_1(c) = 0$ and $h_2(a) = h_2(b) = h_2(c) = 1$. Does this family $\mathcal{H}$ give rise to a universal hash function family?

  Answer: No. E.g., elements a and b collide in both h_1 and h_2.

- Consider the family of all possible hash functions. Is this family universal?

  Answer: Yes, as we saw in class. For all k and k', the probability that we choose a hash function h in the family of all possible hash functions where $h(k) = h(k')$ is $\frac{1}{m}$. This is because, regardless of what $h(k)$ is, the probability that h' maps k to the same value is $\frac{1}{m}$.

- Let n and k be integers greater than 10. Consider the following family $\mathcal{H}$ of hash functions that map elements from the set $\{1, \ldots, n\}^k$ to the set $\{1, \ldots, n\}$. $\mathcal{H}$ contains for every $1 \leq i \leq k$ a function h_i that maps $(x_1, \ldots, x_k) \in \{1, \ldots, n\}^k$ to x_i. Is $\mathcal{H}$ is a universal hash family?

  Answer: $\mathcal{H}$ is not a universal hash family. Two entries from $\{1, \ldots, n\}^k$ that agree on all coordinates but one (e.g. $(1, 1, 1, \ldots 1)$ and $(2, 1, 1, \ldots, 1)$) are mapped to the same element with probability $1 - 1/k > 1/n = 1/m$.

- Let $\mathcal{H}$ be a family of hash functions $h : \mathcal{U} \to \{0, \ldots, m-1\}$ that satisfies the property that for any $h \neq h' \in \mathcal{H}$

$$\Pr_k[h(k) = h'(k)] \leq \frac{1}{m}$$

  Does any universal hash family satisfy this property?

  Answer: No. We know that the set of all possible hash functions is universal. Hence consider this set as our universal family. As our family contains all function, it contains, in particular, the following two function h and h'. The function h is defined as $h(x) = 0$ for all x. The function h' is defined as $h'(x) = 0$ for all $x \neq 0$ and $h'(0) = 1$. Clearly, $\Pr_k[h(k) = h'(k)] = 1 - \frac{1}{u} > \frac{1}{m}$ for an appropriately chosen m and u.

2 Open Addressing

Open addressing is an alternative way (besides hashing with chaining) to resolve collisions. In practice, it can be much faster than hashing with chaining when the load factor $\alpha = \frac{n}{m}$ is small.

An open addressing scheme is defined by a *probing scheme* or function

$$h : \mathcal{U} \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}.$$

This means, for every key k, our probing scheme defines a sequence $h(k, 0), h(k, 1), \ldots, h(k, m-1)$ that we can use to resolve collisions. We use our scheme to implement a hash table as follows:

- **Insert**(k, item): We follow the probing sequence until we find an empty slot and insert the key in the found slot. Note that the special "deleted flag" is treated as an empty slot.
- **Search**(k): We follow the probing sequence until we either find the key or find an empty slot. If the later, we report "NOT FOUND". Note that the "deleted flag" is treated as an occupied slot.
- **Delete**(k): Search for item and replace with a "deleted flag" (i.e. "-1"). Insert treats the flag as an empty slot while search treats it as occupied.

To analyze the performance of open addressing, we typically make the following assumption:

Uniform hashing assumption: The probe sequence of a random key is equally likely to be any of the m! permutations of $\{0, \ldots, m-1\}$.

The assumption means that the probing sequence is as random as possible. Note that the assumption implies that the probing sequence does not repeat slots. This assumption gives us the following nice property:

If the uniform hashing assumption holds, then the expected number of probes in an insert is at most $\frac{1}{1-\alpha}$ *where* $\alpha = \frac{n}{m}$ *is the load factor* (Here n is the number of items in the hash table and m is the number of slots). See the lecture notes for a proof.

In practice, most probing schemes do not satisfy the uniform hashing assumption, but try to come close, so that we can still use it as a tool for analysis.

3 Consistent Hashing

Say that we would like to hash some items to a set of buckets, but buckets may appear or disappear over time. For example, we may want to assign web pages to servers, but servers may either go down or come online. A naive solution is to do normal hashing with m buckets, but when m changes it's possible that the location of all pages may change, which is undesirable.

Consistent Hashing solves this problem by taking some large m ($m = 2^{32}$, for instance), and constructing a hash function h which hashes both the web addresses (items) and names of servers (buckets). Then, page k is assigned to server s, where s is the first server such that $h(s) \geq h(k)$. (where arithmetic is done mod m).

Another way to intuitive way to look at consistent hashing is to consider a ring, and assume that h hashes servers and pages uniformly at random to points on the ring. The

hash values of the n servers divide the ring into n regions; each server is responsible to the region directly to its left. This hashing scheme has the following nice properties:

- Adding and removing servers does not affect most of the web pages.
- Each server gets a fraction $\frac{1}{n}$ of the pages in expectation.

  Proof: Since the hash function is random, due to symmetry a page is equally likely to be stored in each of the n servers. Therefore the probability a page is stored in a given server is $\frac{1}{n}$, so by linearity of expectation this server stores $\frac{1}{n}$ of the pages in expectation.
- With probability at least $1 - \frac{1}{n}$, no server owns more than $\frac{4 \log n}{n}$ of the pages.

  Proof: Divide the ring into $\frac{n}{2 \log n}$ intervals. The probability that the ith server is not hashed into this interval is $1 - \frac{2 \log n}{n}$. Therefore the probability that no server is hashed into this interval is

  $$\left(1 - \frac{2 \log n}{n}\right)^n \leq (e^{-2 \log n/n})^n \leq e^{-2 \log n} = n^{-2}.$$

  Taking the union bound over all intervals, the probability that there exists an interval that does not get a server is at most

  $$n^{-2} \cdot \frac{n}{2 \log n} \leq \frac{1}{n}.$$

  Therefore with probability at least $1 - \frac{1}{n}$ each interval gets a server, which implies that each server is at most an interval of $\frac{4 \log n}{n}$ away from the next server.

4 Perfect Hashing

Static dictionary problem: Given n keys to store in table, we want to support search(k) in worst-case $O(1)$ time. *No insertion or deletion will happen*.

Perfect hashing: [Fredman, Komlós, Szemerédi 1984]

- polynomial build time with high probability (w.h.p.)
- $O(1)$ time for search in *worst* case
- $O(n)$ space in *worst* case

Refer to lecture notes for construction.

5 Practice Problems

5.1 Tight Hashing

Ben Bitdiddle, inspired by the (perfect) hashing lecture, decided to invent his own hashing scheme, which he calls "tight hashing." In this scheme, one hashes the n elements to be stored into an array whose length is exactly n. Assuming that the hash function used is perfectly random, show that, with probability at least $1 - 1/n$, none of the cells in the initial tight hash contain more than $6 \ln n + 1$ elements.

Solution: We will show that, for any *given* cell, the probability that the cell contains more than $K = 6 \ln n + 1$ elements is smaller than $\frac{1}{n^2}$. Then, by union bound, we will have that the probability that *any* cell contains more than K elements is smaller than $n\frac{1}{n^2} = \frac{1}{n}$.

To argue the former, fix a given cell. Let X_i denote the indicator variable of the event that the cell contains element i. Then $X = \sum_i X_i$ represents the number of elements hashed to this cell. We are interested in showing that

$$\Pr[X \geq K] \leq \frac{1}{n^2}$$

We note that $X_1, \ldots, X_n$ are independent $0 - 1$ variables. Therefore, Chernoff bound can be applied.

We start by computing the expected value X, using linearity of expectation on all the X_is.

$$\begin{aligned} E[X] &= E[\sum_i X_i] \\ &= \sum_i E[X_i] \\ &= \sum_i \Pr[X_i = 1] \\ &= \sum_i \frac{1}{n} \\ &= 1 \end{aligned}$$

where the 3^{rd} to 4^{th} line follows from the fact that each element maps to any one of the n cells with equal probability.

Choosing $\beta = 6 \ln n$ and applying version (b) of the Chernoff bound, we get (with $\mu =$

$E[X] = 1$)

$$\begin{aligned}\Pr[X > (1 + 6\ln n)] &= \Pr[X > (1 + \beta)] \\ &\leq e^{-\beta\mu/3} \\ &= e^{-\beta/3} \\ &= e^{-2\ln n} \\ &= \frac{1}{n^2}\end{aligned}$$

5.2 More Perfect Hashing

Alyssa P. Hacker invents her own perfect hashing scheme to hash n items as follows:

- Allocate two arrays A_1 and A_2 of size $2n^{1.5}$ each, where each array is associated with a hash function (h_1 and h_2, respectively).
- Process items in an arbitrary order. For each item x, insert it into $A_1[h_1(x)]$ or $A_2[h_2(x)]$ whichever has fewer items.

Our task is to help Alyssa use this idea to solve the static dictionary problem. (i.e. show that we create an algorithm which allows us to hash all the items with no collisions).

Solution: We break the problem down into steps:

Step 1: Suppose h_1 is sampled from a universal hash family. Fix an arbitrary item i. Show that the probability that $h_1(i) = h_1(j)$ for some other item $j \neq i$ is at most $\frac{1}{2n^{0.5}}$.

Fix an item $j \neq i$. The fact that h_1 is sampled from a universal hash family implies that the probability that i and j are mapped to the same bucket is at most $1/m = 1/2n^{1.5}$. Now, by the union bound, the probability that *there is* an item $j \neq i$ that gets mapped to the same bucket as i is at most $\frac{(n-1)}{2n^{1.5}} < \frac{1}{2n^{0.5}}$.

Step 2: Now suppose that h_1 and h_2 are sampled independently from a universal hash family. Fix an arbitrary item i. Prove that the probability that $h_1(i) = h_1(j)$ for some item $j \neq i$, and that $h_2(i) = h_2(j')$ for some other item $j' \neq i$, is at most $\frac{1}{4n}$.

Since h_1 and h_2 are sampled independently, the probabilities that each one hashes item i to the same bucket as some other item j are also independent. By part (a), the probability of either of these occurring is at most $\frac{1}{2n^{0.5}}$. Hence,

$$\Pr[h_1(i) = h_1(j) \text{ AND } h_2(i) = h_2(j)] = \Pr[h_1(i) = h_1(j)] \Pr[h_1(i) = h_1(j)] \leq \frac{1}{4n}$$

Step 3: Show that, if we pick h_1 and h_2 independently from a universal hash family, the expected number of items i for which there exists $j \neq i$ such that $h_1(i) = h_1(j)$ and there exists $j' \neq i$ such that $h_2(i) = h_2(j')$ is at most $1/4$.

Let X_i be the indicator variable for the event that item i is hashed to buckets with other items. We are looking for $E[\sum_i X_i]$. By linearity of expectations, the above is equal to $\sum_i E[X_i] = \sum_i \Pr[X_i]$. By part (b), for every i, $\Pr[X_i] \leq 1/4n$. Hence, $E[\sum_i X_i] \leq 1/4$.

Step 4: Show how to find, in expected polynomial time, a suitable pair of hash functions that Alyssa can use to solve the static dictionary problem.

This problem is no that different than the scheme discussed earlier for perfect hashing. The particulars are a bit different, but the idea is the same.

Recall the indicator variables X_i defined in part (c). Using Markov's inequality, we know that $\Pr[\sum_i X_i \geq 1] \leq E[\sum_i X_i] \leq 1/4$. Given that all variables are $0, 1$, the probability that one of them is 1 is at most the probability that their sum is more than 1. So, the probability that there exists an item i mapped to the same bucket as other items is at most $1/4$. Or, the probability that every item is mapped to a bucket on its own by at least one of h_1 or h_2 is at least $3/4$. If this happens, clearly the choice of functions works: Every item i is placed in the array with fewer collisions. We know that one of them will be empty and, further, that after i is placed there, no other $j > i$ will be mapped there.

Alyssa can generate a pair of hash functions form a universal hash family, implement the algorithm as described and check that every item i is placed in a bucket with no collisions. If not, then she can repeat until successful. Each attempt can clearly be done in polynomial time.

How many attempts does she need to do until she succeeds? Each attempt is a Bernoulli trial which succeeds with probability at least $3/4$. On average, she needs to try $4/3 < 2$ times before she succeeds. Further, if she repeats $100 \log n$ times, the probability that she hasn't succeeded is $< 1/n^{100}$.