1 Hashing Summary

- Random hash function definition
- Important properties
  - Space
  - Size
  - Properties of r.v $h(x)$
- Truly random hash function
  - Problems with this hash function
- Universal hashing
  - definition
  - Hashing with chaining
- Strong universality
  - definition
  - coordinated sampling
  - lemma 3.2 implication

2 Hashing Detailed

In a hash function we want to map a large set of keys from a universe U into a finite set of values $[m]$. Typically the set of keys will be large. This could be $64 - bit$ integers, for which there are 2^{64} different keys.

2.1 General definition

Definition: A hash function $h : U \rightarrow [m]$ is a random variable in the set of functions $U \rightarrow [m]$

An equivalent definition is that each $h(x), x \in U$ is a random variable. Meaning, either we can view a hash function as a "random" function mapping a typically large set of keys U to a finite set of values $[m]$,or as a random variable $h(x)$ that is defined for each value of $x \in U$. This is the same thing, since picking a value $h(x)$ for each x is the same as a function that takes an x and gives you a value $h(x)$. We typically care about

1. Space - the size of a random seed to calculate $h(x)$
2. Speed - time to calculate $h(x)$
3. Properties of the random variable $h(x)$

2.2 Truly Random Hash function

Using the second (equivalent) definition of a hash function.

Definition: A truly random hash function assigns independently, and uniformly at random, a random variable $h(x)$ to each key $x \in U$.

Since the universe is of size $|U|$, and the we independently and **uniformly** assign a random variable $h(x)$ to each x then, we can choose among $|U|$ different random variables for each x. Since there are $|U|$choices for each $h(x)$, then we must have that the function h is U-dimensional random variable that is picked among all the random hash functions $U \to [m]$. There are $|U|$different hash functions and m different values, meaning we can map the universe keys to values in

$$2^{|U|} \cdot m$$

different ways. Thereby, to store a truly random hash function would require

$$\log_2(2^{|U|}m) = |U|\log_2(2 \cdot m) = |U| + |U|\log_2(m)$$

so we need at least $|U|\log_2 m$ bits to store the hash function. This is way too large to store in ram. We have too much randomness making the hash function large to store. The idea is therefore to make the hash function sufficiently random for the application so that it is fast to store, access and calculate - **the two first properties** important to a hash function.

2.3 Universal hashing

Definition: The hash function h is said to be universal if for any given distinct keys $x, y \in U$, when h is picked uniformly at random we have *low collision*

$$\Pr_h(h(x) = h(y)) \leq \frac{1}{m}$$

The random part here is the hash function h that is chosen over some distribution of functions $U \to [m]$.

2.3.1 Hashing with chaining

We can use hashing to construct *hash tables with chaining*. That is, if we have a set of keys $S \subseteq U$ we wish to store, then we wish to be able to lookup any key in S in expected constant time - this is the hash table. We let $n = |S|$ and pick $m \geq n$ and a universal hash function $h : U \to [m]$. Now we construct our hash table with chaining as follows. We create an array L of length m. Each element $L[i]$ is a linked lists(also known as chain) for each key that *hashes* to $i \in [m]$. The important thing about the hash table was that we wish to lookup an element in expected constant time. We can execute the lookup operation by checking whether a key $x \in U$ is in the hash table L by the lookup $L[h(x)]$, since h is a function $U \to [m]$ and we created L to have length m. This lookup takes time proportional to $1 + |L[h(x)]|$. We add the one, since the lookup into

L is always one operation even when nothing is there. Thereby, the lookup time of our hash table is bounded by the quality of the hash function, since this determines the number of expected collisions.

Theorem: for $x \notin S$ and a universal hash function h we have $E[|L[h(x)]|] \leq 1$.

Proof: The length of the linked list $L[h(x)] = L[i]$ is the number of keys $y \in S$ that also hash to $h(y) = i$ entrance i. We take the expectation with respect to the random variable which is the hash function h

$$E_h[|L[h(x)|] = E_h \left[\sum_{y \in S} [h(x) = h(y)] \right]$$

where the equality holds due to the above argument and $[h(x) = h(y)]$ counts exactly the number of times that $h(y)$ hashes to the same bucket as $h(x)$ and we thereby have a collision. Continuing we get

$$= \sum_{y \in S} E_h \left[[h(x) = h(y)] \right] = \sum_{y \in S} \Pr_h \left([h(x) = h(y)] \right) \leq \sum_{y \in S} \frac{1}{m} = \frac{n}{m} \leq 1$$

where the first inequality holds since we use Iverson bracket notation, and the expectation of an indicator variable is the probability of success.

2.3.2 Signatures

2.4 Strong universality

In *strong universality* we consider a *random* hash function $h : U \to [m]$ and pairwise events: given distinct keys $x, y \in U$ and two possibly non-distinct values $q, r \in [m]$.

Definition: The hash function $h : U \to [m]$ is said to be strongly universal if the probability for every pairwise event $\Pr_h(h(x) = q \wedge h(y) = r) \leq \frac{1}{m^2}$

An equivalent definition is that each key is hashed uniformly into $[m]$ and buckets, and every two distinct keys are hashed independently.

We note that strongly universal still implies universal hashing. Consider a strongly universal hash function, and let us look at the definition for strong universality

$$\Pr\left[h(x) = h(y)\right] =$$

2.4.1 Coordinated sampling

Let us consider sampling from the subset $A \subseteq U$. We sample using a strongly universal hash function h and a threshold $t \in \{0, 1, ..., m\}$ such that a sample is picked if $h(x) < t$. Since $h(x)$ is hashed uniformly at random(h is strongly universal and we can use the equivalent definition), then when we fix t, the probability of choosing a sample is $\frac{t}{m}$(there is m ways of hashing x and t of which we want). Now we consider what we sample from the subset A given our hash function h and threshold t. Let $S_{h,t} = \{x \in A | h(x) < t\}$ be the samples we take from the subset A. The expectation of the size of the set is

$$E_h\left[|S_{h,t}(A)|\right] = E_h\left[\sum_{y \in A} \left[\Pr\left(h(y) < t\right)\right]\right] = \sum_{y \in A} E\left[\left[\Pr\left(h(y) < t\right)\right]\right] = \sum_{y \in A} \frac{t}{m} = |A|\frac{t}{m}$$

so we can actually estimate the size of $|A|$ if we have the sample $S_{h,t}$ by $|S_{h,t}| \cdot \frac{m}{t}$.

Universal sampling alone is however not enough. Multiplication-shift-scheme will for instance have that $h(x) = 0$, and therefore we will always sample 0 in case that $t > 0$.

The important part is however sampling from different sets B and C and comparing their similarities by $B \cup C$ and $B \cap C$. Suppose, we have the samples $S_{h,t}(B)$ and $S_{h,t}(C)$. We can calculate the sample union and sample intersection as

$$S_{h,t}(B \cup C) = S_{h,t}(B) \cup S_{h,t}(C)$$

$$S_{h,t}(B \cap C) = S_{h,t}(B) \cap S_{h,t}(C)$$

As was shown earlier, we can then for instance calculate the size $|B \cap C| = \frac{m}{t} \cdot |S_{h,t}(B \cap C)|$. This means we can sample from different sets in a distributed setting, as long as each agent shares the same hash function and threshold. For instance, in machine learning, we can sample from a new set to figure out which previously recorded sample set it has most in common with. Another thing to note, is that when we talk about a set A now, it could as well be represented as a union or an intersection of sets(meaning we could do it in a coordinated fashion).

The sampling probability we got when fixing the threshold t was $\frac{t}{m}$required only that the hash function h was uniform. This was also used to show that $E_h\left[|S_{h,t}(A)|\right] = |A|\frac{t}{m}$ and we can get an unbiased estimate of the size of A. We could actually say $E_h\left[|S_{h,t}(A)| \cdot \frac{m}{t}\right] = \frac{m}{t} \cdot E_h\left[|S_{h,t}(A)|\right] = |A|$. **Lemma 3.2** requires pairwise independence. It says that given $X = \sum_{a \in A} X_a$where $X_a$are pairwise independent($X_a$is the same as $|S_{h,t}(A)|$) with mean $E[X] = \mu$ then $Var(X) \leq \mu$ and for any $q > 0$

$$\Pr\left[|X - \mu| \geq q\sqrt{\mu}\right] \leq \frac{1}{q^2}$$

thereby we can bound how close the estimate is to the mean(how good is $|A|\frac{t}{m}$) , and thereby how good our estimate of $|A|$is. That is, we can show how concentrated around the mean are estimate is. The closer the tighter the estimate.