Universal and Perfect Hashing

Lecture 10

September 26, 2019

Announcements and Overview

- Pset 4 released and due on Thursday, October 3 at 10am. Note one day extension over usual deadline.
- Midterm 1 is on Monday, Oct 7th from 7-9.30pm. More details and conflict exam information will be posted on Piazza.
- Next pset will be released after the midterm exam.

Announcements and Overview

- Pset 4 released and due on Thursday, October 3 at 10am. Note one day extension over usual deadline.
- Midterm 1 is on Monday, Oct 7th from 7-9.30pm. More details and conflict exam information will be posted on Piazza.
- Next pset will be released after the midterm exam.

Today's lecture:

- Review pairwise independence and related constructions
- (Strongly) Universal hashing
- Perfect hashing

Part I

Review

Pairwise independent random variables

Definition

Random variables $X_1, X_2, \ldots, X_n$ from a range B are **pairwise** independent if for all $1 \leq i < j \leq n$ and for all $b, b' \in B$,

$$\Pr[X_i = b, X_j = b'] = \Pr[X_i = b] \cdot \Pr[X_j = b'] .$$

Constructing pairwise independent rvs

Suppose we want to create n pairwise independent random variables in range $0, 1, \ldots, m-1$. That is we want to generate $X_0, X_2, \ldots, X_{n-1}$ such that

- $\Pr[X_i = \alpha] = 1/m$ for each $\alpha \in \{0, 1, 2, \ldots, m-1\}$
- X_i and X_j are independent for any $i \neq j$

Constructing pairwise independent rvs

Suppose we want to create n pairwise independent random variables in range $0, 1, \ldots, m-1$. That is we want to generate $X_0, X_2, \ldots, X_{n-1}$ such that

- $\Pr[X_i = \alpha] = 1/m$ for each $\alpha \in \{0, 1, 2, \ldots, m-1\}$
- X_i and X_j are independent for any $i \neq j$

Interesting case: $n = m = p$ where p is a prime number

- Pick a, b uniformly at random from $\{0, 1, 2, \ldots, p-1\}$
- Set $X_i = ai + b$
- Only need to store a, b. Can generate X_i from i.

Constructing pairwise independent rvs

Suppose we want to create n pairwise independent random variables in range $0, 1, \ldots, m-1$. That is we want to generate $X_0, X_2, \ldots, X_{n-1}$ such that

- $\Pr[X_i = \alpha] = 1/m$ for each $\alpha \in \{0, 1, 2, \ldots, m-1\}$
- X_i and X_j are independent for any $i \neq j$

Interesting case: $n = m = p$ where p is a prime number

- Pick a, b uniformly at random from $\{0, 1, 2, \ldots, p-1\}$
- Set $X_i = ai + b$
- Only need to store a, b. Can generate X_i from i.

Relies on the fact that $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ is a field

Pairwise independence for general n and m

A rough sketch.
If $n < m$ we can use a prime $p \in [m, 2m]$ (one always exists) and use the previous construction based on $\mathbb{Z}_p$.

$n > m$ is the more difficult case and also relevant.

The following is a fundamental theorem on finite fields.

Theorem

Every finite field $\mathbb{F}$ has order p^k for some prime p and some integer $k \geq 1$. For every prime p and integer $k \geq 1$ there is a finite field $\mathbb{F}$ of order p^k and is unique up to isomorphism.

We will assume n and m are powers of 2. From above can assume we have a field $\mathbb{F}$ of size $n = 2^k$.

Pairwise independence when n, m are powers of 2

We will assume n and m are powers of 2.
We have a field $\mathbb{F}$ of size $n = 2^k$.

Generate n pairwise independent random variables from $[n]$ to $[n]$ by picking random $a, b \in \mathbb{F}$ and setting $X_i = ai + b$ (operations in $\mathbb{F}$). From previous proof $X_1, \ldots, X_n$ are pairwise independent.

Now $X_i \in [n]$. Truncate X_i to $[m]$ by dropping the most significant $\log n - \log m$ bits. Resulting variables are still pairwise independent (both n, m being powers of 2 important here).

Skipping details on computational aspects of $\mathbb{F}$ which are closely tied to the proof of the theorem on fields.

Pairwise Independence and Chebyshev's Inequality

Chebyshev's Inequality

For $a \geq 0$, $\Pr[|X - \mathbf{E}[X]| \geq a] \leq \frac{Var(X)}{a^2}$ equivalently for any $t > 0$, $\Pr[|X - \mathbf{E}[X]| \geq t\sigma_X] \leq \frac{1}{t^2}$ where $\sigma_X = \sqrt{Var(X)}$ is the standard deviation of X.

Suppose $X = X_1 + X_2 + \ldots + X_n$.
If $X_1, X_2, \ldots, X_n$ are independent then $Var(X) = \sum_i Var(X_i)$.

Pairwise Independence and Chebyshev's Inequality

Chebyshev's Inequality

For $a \geq 0$, $\Pr[|X - \mathbf{E}[X]| \geq a] \leq \frac{Var(X)}{a^2}$ equivalently for any $t > 0$, $\Pr[|X - \mathbf{E}[X]| \geq t\sigma_X] \leq \frac{1}{t^2}$ where $\sigma_X = \sqrt{Var(X)}$ is the standard deviation of X.

Suppose $X = X_1 + X_2 + \ldots + X_n$.
If $X_1, X_2, \ldots, X_n$ are independent then $Var(X) = \sum_i Var(X_i)$.

Lemma

Suppose $X = \sum_i X_i$ and $X_1, X_2, \ldots, X_n$ are pairwise independent, then $Var(X) = \sum_i Var(X_i)$.

Hence pairwise independence suffices if one relies only on Chebyshev inequality.

Part II

Hash Tables

Dictionary Data Structure

1. $\mathcal{U}$: universe of keys with total order: numbers, strings, etc.
2. Data structure to store a subset $S \subseteq \mathcal{U}$
3. **Operations:**
   1. **Search/look up**: given $x \in \mathcal{U}$ is $x \in S$?
   2. **Insert**: given $x \notin S$ add x to S.
   3. **Delete**: given $x \in S$ delete x from S
4. **Static** structure: S given in advance or changes very infrequently, main operations are lookups.
5. **Dynamic** structure: S changes rapidly so inserts and deletes as important as lookups.

Can we do everything in $O(1)$ time?

Hashing and Hash Tables

Hash Table data structure:

1. A (hash) table/array T of size m (the table **size**).
2. A hash function $h : \mathcal{U} \rightarrow \{0, \ldots, m-1\}$.
3. Item $x \in \mathcal{U}$ hashes to slot $h(x)$ in T.

Hashing and Hash Tables

Hash Table data structure:

1. A (hash) table/array T of size m (the table **size**).
2. A hash function $h : \mathcal{U} \rightarrow \{0, \ldots, m-1\}$.
3. Item $x \in \mathcal{U}$ hashes to slot $h(x)$ in T.

Given $S \subseteq \mathcal{U}$. How do we store S and how do we do lookups?

Ideal situation:

1. Each element $x \in S$ hashes to a distinct slot in T. Store x in slot $h(x)$
2. **Lookup**: Given $y \in \mathcal{U}$ check if $T[h(y)] = y$. $O(1)$ time!

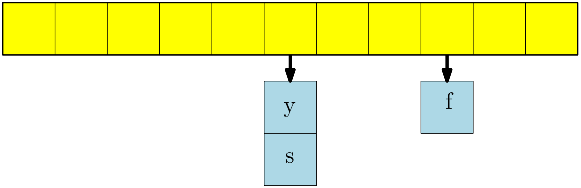
Collisions unavoidable if $|T| < |\mathcal{U}|$.

Handling Collisions: Chaining

Collision: $h(x) = h(y)$ for some $x \neq y$.

Chaining/Open hashing to handle collisions:

1. For each slot i store all items hashed to slot i in a linked list. $T[i]$ points to the linked list
2. **Lookup**: to find if $y \in \mathcal{U}$ is in T, check the linked list at $T[h(y)]$. Time proportion to size of linked list.



Does hashing give $O(1)$ time per operation for dictionaries?

Hash Functions

Parameters: $N = |\mathcal{U}|$ (very large), $m = |T|$, $n = |S|$
Goal: $O(1)$-time lookup, insertion, deletion.

Single hash function

If $N \geq m^2$, then for any hash function $h : \mathcal{U} \to T$ there exists $i < m$ such that at least $N/m \geq m$ elements of $\mathcal{U}$ get hashed to slot i. Any S containing all of these is a **very very bad set for h**!

Such a bad set may lead to $O(m)$ lookup time!

In practice:

- Dictionary applications: choose a simple hash function and hope that worst-case bad sets do not arise
- Crypto applications: create "hard" and "complex" function very carefully which makes finding collisions difficult

Hashing from a theoretical point of view

- Consider a family $\mathcal{H}$ of hash functions with *good properties* and choose h randomly from $\mathcal{H}$
- Guarantees: small # collisions in expectation for any given S.
- $\mathcal{H}$ should allow efficient sampling.
- Each $h \in \mathcal{H}$ should be efficient to evaluate and require small memory to store.

In other words a hash function is a "pseudorandom" function

Strongly Universal Hashing

1. **Uniform:** Consider any element $x \in \mathcal{U}$. Then if $h \in \mathcal{H}$ is picked randomly then x should go into a random slot in T. In other words $\Pr[h(x) = i] = 1/m$ for every $0 \le i < m$.
2. **(2)-Strongly Universal**: Consider any two distinct elements $x, y \in \mathcal{U}$. Then if $h \in \mathcal{H}$ is picked randomly then $h(x)$ and $h(y)$ should be independent random variables.

Universal Hashing

- **(2)-Universal:** Consider any two distinct elements $x, y \in \mathcal{U}$. Then if $h \in \mathcal{H}$ is picked randomly then the probability of a collision between x and y should be at most $1/m$. In other words $\Pr[h(x) = h(y)] \leq 1/m$.

Note: we do not insist on uniformity.

Universal Hashing

- **(2)-Universal:** Consider any two distinct elements $x, y \in \mathcal{U}$. Then if $h \in \mathcal{H}$ is picked randomly then the probability of a collision between x and y should be at most $1/m$. In other words $\Pr[h(x) = h(y)] \leq 1/m$.

Note: we do not insist on uniformity.

Universal hashing is a relaxation of strong universal hashing and simpler to construct while retaining most of the useful properties.

(Strongly) Universal Hashing

Definition

A family of hash functions $\mathcal{H}$ is (2-)**strongly universal** if for all distinct $x, y \in \mathcal{U}$, $h(x)$ and $h(y)$ are independent for h chosen uniformly at random from $\mathcal{H}$, and for all x, $h(x)$ is uniformly distributed.

Definition

A family of hash functions $\mathcal{H}$ is (2-)**universal** if for all distinct $x, y \in \mathcal{U}$, $\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] \leq 1/m$ where m is the table size.

Analyzing Universal Hashing

1. T is hash table of size m.
2. $S \subseteq \mathcal{U}$ is a **fixed** set of size n
3. h is chosen randomly from a universal hash family $\mathcal{H}$.
4. x is a *fixed* element of $\mathcal{U}$.

Question: What is the *expected* time to look up x in T using h assuming chaining used to resolve collisions?

Analyzing Universal Hashing

1. T is hash table of size m.
2. $S \subseteq \mathcal{U}$ is a **fixed** set of size n
3. h is chosen randomly from a universal hash family $\mathcal{H}$.
4. x is a *fixed* element of $\mathcal{U}$.

Question: What is the *expected* time to look up x in T using h assuming chaining used to resolve collisions?

1. The time to look up x is the size of the list at $T[h(x)]$: same as the number of elements in S that collide with x under h.
2. $\ell(x)$ be this number. We want $\mathrm{E}[\ell(x)]$
3. Let $C_{x,y}$ be indicator random variable for x, y colloding under h, that $C_{x,y} = 1$ iff $h(x) = h(y)$

Analyzing Universal Hashing

Continued...

Number of elements colliding with x: $\ell(x) = \sum_{y \in S} C_{x,y}$.

$$
\begin{aligned}
\Rightarrow \mathrm{E}[\ell(x)] &= \sum_{y \in S, y \neq x} \mathrm{E}[C_{x,y}] && \text{linearity of expectation} \\
&= \sum_{y \in S, y \neq x} Pr[h(x) = h(y)] \\
&\leq \sum_{y \in S, y \neq x} \frac{1}{m} && (\text{since } \mathcal{H} \text{ is a universal hash family}) \\
&\leq |S|/m \\
&\leq \frac{n}{m} \\
&\leq 1 && (\text{if } |S| \leq m)
\end{aligned}
$$

Analyzing Universal Hashing

Comments:

1. Expected time for insertion and deletion also $O(1)$ if $n \leq m$.
2. Analysis assumes static set S but holds as long as S is a set formed with at most $O(m)$ insertions and deletions. Assumption is that insertions and deletions are not adaptive.
3. **Worst-case**: look up time can be large! How large? Technically $O(n)$ if all elements collide.

Analyzing Universal Hashing: Maximum Load

If h is a fully random function and $m = n$ then expected maximum load in any bucket of T is $O(\log n / \log\log n)$ via balls and bin analogy.

If h is chosen from a universal hash family $\mathcal{H}$ what is the expected maximum load?

Lemma

Let h be chosen from a universal hash family and let $m \geq n$ and let L be maximum load of any slot. Then $\Pr\left[L > t\sqrt{n}\right] \leq 1/t^2$ for $t \geq 1$.

Analyzing Universal Hashing: Maximum Load

If h is a fully random function and $m = n$ then expected maximum load in any bucket of T is $O(\log n / \log \log n)$ via balls and bin analogy.

If h is chosen from a universal hash family $\mathcal{H}$ what is the expected maximum load?

Lemma

Let h be chosen from a universal hash family and let $m \geq n$ and let L be maximum load of any slot. Then $\Pr\left[L > t\sqrt{n}\right] \leq 1/t^2$ for $t \geq 1$.

Thus $L = O(\sqrt{n})$ with probability at least $1/2$.

Analyzing Universal Hashing: Maximum Load

Lemma

Let h be chosen from a universal hash family and let $m \geq n$ and let L be maximum load of any slot. Then $\Pr[L > t\sqrt{n}] \leq 1/t^2$ for $t \geq 1$.

Let $C = \sum_{x,y \in S, x \neq y} C_{x,y}$ be total number of collisions.

- $\mathbf{E}[C] \leq \binom{n}{2}/m \leq (n-1)/2$ if $m \geq n$.
- **Observation:** $C \geq \binom{L}{2}$. Why?
- $L > t\sqrt{n}$ implies $C > t^2 n/2$.
- By Markov $\Pr[C > t^2 n/2] \leq \mathbf{E}[C]/(t^2 n/2) \leq 1/t^2$
- Hence $\Pr[L > t\sqrt{n}] \leq 1/t^2$.

Analyzing Universal Hashing: Maximum Load

Lemma

Let h be chosen from a universal hash family and let $m \geq n$ and let L be maximum load of any slot. Then $\mathbf{E}[L] = O(\sqrt{n})$.

Direct proof: $(\mathbf{E}[L])^2 \leq \mathbf{E}\left[L^2\right] \leq \mathbf{E}[C] \leq n$ (using Jensen's ineq)

L is a non-negative random variable in range. Hence

$$
\begin{aligned}
E[L] &= \sum_{i=1}^{n} \Pr[L \geq i] \quad \text{(from defn of expectation)} \\
&\leq \sum_{i=1}^{\sqrt{n}} 1 + \sum_{i=\sqrt{n}+1}^{n} n/i^2 \quad \text{(from previous lemma)} \\
&\leq \sqrt{n} + n \int_{\sqrt{n}}^{n} 1/i^2 \leq 2\sqrt{n}.
\end{aligned}
$$

Compact Strongly Universal Hash Family

Parameters: $N = |\mathcal{U}|$, $m = |T|$, $n = |S|$

Question: How do we construct strongly universal hash family?

Compact Strongly Universal Hash Family

Parameters: $N = |\mathcal{U}|$, $m = |T|$, $n = |S|$

Question: How do we construct strongly universal hash family?

If N and m are powers of 2 then use construction of N pairwise independent random variables over range $[m]$ discussed previously

Compact Strongly Universal Hash Family

Parameters: $N = |\mathcal{U}|$, $m = |T|$, $n = |S|$

Question: How do we construct strongly universal hash family?

If N and m are powers of 2 then use construction of N pairwise independent random variables over range $[m]$ discussed previously

Disadvantage: Need m to be power of 2 and requires complicated field operations

Compact Universal Hash Family

Parameters: $N = |\mathcal{U}|$, $m = |T|$, $n = |S|$

1. Choose a **prime** number $p > N$. Define function $h_{a,b}(x) = ((ax + b) \mod p) \mod m$.
2. Let $\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$ ($\mathbb{Z}_p = \{0, 1, \ldots, p - 1\}$). Note that $|\mathcal{H}| = p(p - 1)$.

Compact Universal Hash Family

Parameters: $N = |\mathcal{U}|$, $m = |T|$, $n = |S|$

1. Choose a **prime** number $p > N$. Define function $h_{a,b}(x) = ((ax + b) \mod p) \mod m$.
2. Let $\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$ ($\mathbb{Z}_p = \{0, 1, \ldots, p - 1\}$). Note that $|\mathcal{H}| = p(p - 1)$.

Theorem

$\mathcal{H}$ is a universal hash family.

Compact Universal Hash Family

Parameters: $N = |\mathcal{U}|$, $m = |T|$, $n = |S|$

1. Choose a **prime** number $p > N$. Define function $h_{a,b}(x) = ((ax + b) \mod p) \mod m$.
2. Let $\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$ ($\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$). Note that $|\mathcal{H}| = p(p-1)$.

Theorem

$\mathcal{H}$ is a universal hash family.

Comments:

1. $h_{a,b}$ can be evaluated in $O(1)$ time.
2. Easy to store, *i.e.*, just store a, b. Easy to sample.

Understanding the hashing

Once we fix a and b, and we are given a value x, we compute the hash value of x in two stages:

1. **Compute**: $r \leftarrow (ax + b) \mod p$.
2. **Fold**: $r' \leftarrow r \mod m$

Let $g_{a,b}(x) = (ax + b) \mod p$.
$h_{a,b}(x) = g_{a,b}(x) \mod m$.

Understanding the hashing

Once we fix a and b, and we are given a value x, we compute the hash value of x in two stages:

1. **Compute**: $r \leftarrow (ax + b) \mod p$.
2. **Fold**: $r' \leftarrow r \mod m$

Let $g_{a,b}(x) = (ax + b) \mod p$.
$h_{a,b}(x) = g_{a,b}(x) \mod m$.

Fix x:

- $g_{a,b}(x)$ is uniformly distributed in $\{0, 1, \ldots, p - 1\}$. Why?
- However $h_{a,b}(x)$ is not necessarily uniformly distributed over $\{0, 1, 2, \ldots, m\}$. Why?

Some math required...

Recall $\mathbb{Z}_p$ is a field.

- $a \neq 0$ implies unique a' such that $aa' = 1 \mod p$
- For $a, x, y \in \mathbb{Z}_p$ such that $x \neq y$ and $a \neq 0$ we have $ax \neq ay \mod p$.
- For $x \neq y$ and any r, s there is a unique solution (a, b) to the equations $ax + b = r$ and $ay + b = s$.

Proof of the Theorem: Outline

$h_{a,b}(x) = ((ax + b) \mod p) \mod m)$.

Theorem

$\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$ *is universal.*

Proof.

Fix $x, y \in \mathcal{U}, x \neq y$. Show that
$\Pr_{h_{a,b} \sim \mathcal{H}}[h_{a,b}(x) = h_{a,b}(y)] \leq 1/m$.
Note that $|\mathcal{H}| = p(p-1)$.

Proof of the Theorem: Outline

$h_{a,b}(x) = ((ax + b) \mod p) \mod m)$.

Theorem

$\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$ *is universal.*

Proof.

Fix $x, y \in \mathcal{U}, x \neq y$. Show that
$\Pr_{h_{a,b} \sim \mathcal{H}}[h_{a,b}(x) = h_{a,b}(y)] \leq 1/m$.
Note that $|\mathcal{H}| = p(p-1)$.

1. Let (a, b) (equivalently $h_{a,b}$) be *bad* for x, y if $h_{a,b}(x) = h_{a,b}(y)$.

Proof of the Theorem: Outline

$h_{a,b}(x) = ((ax + b) \mod p) \mod m)$.

Theorem

$\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$ *is universal.*

Proof.

Fix $x, y \in \mathcal{U}, x \neq y$. Show that
$\Pr_{h_{a,b} \sim \mathcal{H}}[h_{a,b}(x) = h_{a,b}(y)] \leq 1/m$.
Note that $|\mathcal{H}| = p(p-1)$.

1. Let (a, b) (equivalently $h_{a,b}$) be *bad* for x, y if $h_{a,b}(x) = h_{a,b}(y)$.
2. **Claim:** Number of bad (a, b) is at most $p(p-1)/m$.

Proof of the Theorem: Outline

$h_{a,b}(x) = ((ax + b) \mod p) \mod m)$.

Theorem

$\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$ *is universal.*

Proof.

Fix $x, y \in \mathcal{U}, x \neq y$. Show that
$\Pr_{h_{a,b} \sim \mathcal{H}}[h_{a,b}(x) = h_{a,b}(y)] \leq 1/m$.
Note that $|\mathcal{H}| = p(p-1)$.

1. Let (a, b) (equivalently $h_{a,b}$) be *bad* for x, y if $h_{a,b}(x) = h_{a,b}(y)$.
2. **Claim:** Number of bad (a, b) is at most $p(p-1)/m$.
3. Total number of hash functions is $p(p-1)$ and hence probability of a collision is $\leq 1/m$.

□

Proof of Claim

$$h_{a,b}(x) = (((ax + b) \mod p) \mod m)$$

2 lemmas ...

Fix $x \neq y \in \mathbb{Z}_p$, and let $r = (ax + b)\ mod\ p$ and $s = (ay + b)\ mod\ p$.

Proof of Claim

$$h_{a,b}(x) = (((ax + b) \mod p) \mod m)$$

2 lemmas ...

Fix $x \neq y \in \mathbb{Z}_p$, and let $r = (ax + b)\ mod\ p$ and $s = (ay + b)\ mod\ p$.

1. 1-to-1 correspondence between $p(p-1)$ pairs of (a, b) (equivalently $h_{a,b}$) and $p(p-1)$ pairs of (r, s).

Proof of Claim

$$h_{a,b}(x) = (((ax + b) \mod p) \mod m)$$

2 lemmas ...

Fix $x \neq y \in \mathbb{Z}_p$, and let $r = (ax + b)\ mod\ p$ and $s = (ay + b)\ mod\ p$.

1. 1-to-1 correspondence between $p(p-1)$ pairs of (a, b) (equivalently $h_{a,b}$) and $p(p-1)$ pairs of (r, s).

2. Out of all possible $p(p-1)$ pairs of (r, s), at most $p(p-1)/m$ fraction satisfies $r\ mod\ m = s\ mod\ m$.

Correspondence Lemma

Lemma

If $x \neq y$ *then for each* (r, s) *such that* $r \neq s$ *and* $0 \leq r, s \leq p-1$ *there is exactly* **one** *pair* (a, b) *such that* $a \neq 0$ *and* $ax + b \mod p = r$ *and* $ay + b \mod p = s$.

Proof.

Solve the two equations:

$$ax + b = r \mod p \qquad \text{and} \qquad ay + b = s \mod p$$

We get $a = \frac{r-s}{x-y} \mod p$ and $b = r - ax \mod p$. □

One-to-one correspondence between (a, b) and (r, s)

Collisions due to folding

Once we fix a and b, and we are given a value x, we compute the hash value of x in two stages:

1. **Compute**: $r \leftarrow (ax + b) \mod p$.
2. **Fold**: $r' \leftarrow r \mod m$

Collision...

Given two distinct values x and y they might collide only because of folding.

Collisions due to folding

Once we fix a and b, and we are given a value x, we compute the hash value of x in two stages:

1. **Compute**: $r \leftarrow (ax + b) \mod p$.
2. **Fold**: $r' \leftarrow r \mod m$

Collision...

Given two distinct values x and y they might collide only because of folding.

Lemma

$\#$ of pairs (r, s) of $\mathbb{Z}_p \times \mathbb{Z}_p$ such that $r \neq s$ and $r \mod m = s \mod m$ is at most $p(p-1)/m$.

Folding numbers

Lemma

 # pairs $(r, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$ *such that* $r \neq s$ *and* $r \mod m = s \mod m$ *(folded to the same number) is* $p(p-1)/m$.

Proof.

Consider a pair $(r, s) \in \{0, 1, \ldots, p-1\}^2$ s.t. $r \neq s$. Fix r:

1. Let $d = r \mod m$.

Folding numbers

Lemma

 # pairs $(r, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$ *such that* $r \neq s$ *and* $r \mod m = s \mod m$ *(folded to the same number) is* $p(p-1)/m$.

Proof.

Consider a pair $(r, s) \in \{0, 1, \ldots, p-1\}^2$ s.t. $r \neq s$. Fix r:

1. Let $d = r \mod m$.
2. There are $\lceil p/m \rceil$ values of s such that $r \mod m = s \mod m$.
3. One of them is when $r = s$.
4. $\implies$ # of colliding pairs

Folding numbers

Lemma

$\#$ pairs $(r,s) \in \mathbb{Z}_p \times \mathbb{Z}_p$ such that $r \neq s$ and $r \mod m = s \mod m$ (folded to the same number) is $p(p-1)/m$.

Proof.

Consider a pair $(r,s) \in \{0,1,\ldots,p-1\}^2$ s.t. $r \neq s$. Fix r:

1. Let $d = r \mod m$.
2. There are $\lceil p/m \rceil$ values of s such that $r \mod m = s \mod m$.
3. One of them is when $r = s$.
4. $\implies$ $\#$ of colliding pairs $(\lceil p/m \rceil - 1)p \leq (p-1)p/m$

□

Proof of Claim

of bad pairs is $\mathbf{p(p-1)/m}$

Proof.

Let $a, b \in \mathbb{Z}_p$ such that $a \neq 0$ and $h_{a,b}(x) = h_{a,b}(y)$.

1. Let $r = ax + b \mod p$ and $s = ay + b \mod p$.
2. Collision if and only if $r \mod m = s \mod m$.
3. (Folding error): Number of pairs (r, s) such that $r \neq s$ and $0 \leq r, s \leq p - 1$ and $r \mod m = s \mod m$ is $p(p-1)/m$.
4. From previous lemma there is one-to-one correspondence between (a, b) and (r, s). Hence total number of bad (a, b) pairs is $p(p-1)/m$.

□

Proof of Claim

of bad pairs is $\mathbf{p(p-1)/m}$

Proof.

Let $a, b \in \mathbb{Z}_p$ such that $a \neq 0$ and $h_{a,b}(x) = h_{a,b}(y)$.

1. Let $r = ax + b \mod p$ and $s = ay + b \mod p$.
2. Collision if and only if $r \mod m = s \mod m$.
3. (Folding error): Number of pairs (r, s) such that $r \neq s$ and $0 \leq r, s \leq p - 1$ and $r \mod m = s \mod m$ is $p(p-1)/m$.
4. From previous lemma there is one-to-one correspondence between (a, b) and (r, s). Hence total number of bad (a, b) pairs is $p(p-1)/m$.

□

Prob of x and y to collide: $\frac{\#\text{ bad } (a,b) \text{ pairs}}{\#(a,b) \text{ pairs}} = \frac{p(p-1)/m}{p(p-1)} = \frac{1}{m}$.

Part III

Perfect Hashing

Perfect Hashing

Question: Suppose we get a set $S \subset \mathcal{U}$ of size n. Can we design an "efficient" and "perfect" hash function?

- Create a table T of size $m = O(n)$.
- Create a hash function $h : S \to [m]$ with no collisions!
- h should be fast and efficient to evaluate
- Construct h efficiently given S. Construction of h can be randomized (Las Vegas algorithm)

Perfect Hashing

Question: Suppose we get a set $S \subset \mathcal{U}$ of size n. Can we design an "efficient" and "perfect" hash function?

- Create a table T of size $m = O(n)$.
- Create a hash function $h : S \to [m]$ with no collisions!
- h should be fast and efficient to evaluate
- Construct h efficiently given S. Construction of h can be randomized (Las Vegas algorithm)

A perfect hash function would guarantee lookup time of $O(1)$.

Perfect Hashing via Large Space

Suppose $m = n^2$. Table size is much bigger than n

Lemma

Suppose $\mathcal{H}$ is a universal hash family and $m = n^2$. Then $\Pr_{h \in \mathcal{H}}[\textit{no collisions in } S] \geq 1/2$.

Perfect Hashing via Large Space

Suppose $m = n^2$. Table size is much bigger than n

Lemma

Suppose $\mathcal{H}$ is a universal hash family and $m = n^2$. Then $\Pr_{h \in \mathcal{H}}[\text{no collisions in } S] \geq 1/2$.

Proof.

- Total number of collisions is $C = \sum_{x,y \in S, x \neq y} C_{x,y}$.
- $\mathbf{E}[C] \leq \binom{n}{2}/m < 1/2$.
- By Markov inequality $\Pr[C \geq 1] < 1/2$.

□

Perfect Hashing via Large Space

Suppose $m = n^2$. Table size is much bigger than n

Lemma

Suppose $\mathcal{H}$ is a universal hash family and $m = n^2$. Then $\Pr_{h \in \mathcal{H}}[\text{no collisions in } S] \geq 1/2$.

Proof.

- Total number of collisions is $C = \sum_{x,y \in S, x \neq y} C_{x,y}$.
- $\mathbf{E}[C] \leq \binom{n}{2}/m < 1/2$.
- By Markov inequality $\Pr[C \geq 1] < 1/2$.

□

Algorithm: pick $h \in \mathcal{H}$ randomly and check if h is perfect. Repeat until success.

Perfect Hashing

Two levels of hash tables

Question: Can we obtain perfect hashing with $m = O(n)$?

Perfect Hashing

- Do hashing once with table T of size m
- For each slot i in T let Y_i be number of elements hashed to slot i. If $Y_i > 1$ use perfect hashing with second table T_i of size Y_i^2.

Perfect Hashing

Two levels of hash tables

Question: Can we obtain perfect hashing with $m = O(n)$?

Perfect Hashing

- Do hashing once with table T of size m
- For each slot i in T let Y_i be number of elements hashed to slot i. If $Y_i > 1$ use perfect hashing with second table T_i of size Y_i^2.

Construction gives perfect hashing. What is the space used?

Perfect Hashing

Two levels of hash tables

Question: Can we obtain perfect hashing with $m = O(n)$?

Perfect Hashing

- Do hashing once with table T of size m
- For each slot i in T let Y_i be number of elements hashed to slot i. If $Y_i > 1$ use perfect hashing with second table T_i of size Y_i^2.

Construction gives perfect hashing. What is the space used?

$$Z = m + \sum_{i=0}^{m-1} Y_i^2$$

a random variable (depends on random choice of first level hash function)

Perfect Hashing
O(n) space usage

h the primary random hash function.

Perfect Hashing

O(n) space usage

h the primary random hash function.

Claim

$\mathbf{E}\left[\sum_{i=0}^{m-1} Y_i^2\right] \leq 3n/2$ *if* $m \geq n$.

Perfect Hashing

O(n) space usage

h the primary random hash function.

Claim

$\mathbf{E}\left[\sum_{i=0}^{m-1} Y_i^2\right] \leq 3n/2$ *if* $m \geq n$.

Proof.

Let C be total number of collisions. We already saw $\mathbf{E}[C] \leq \binom{n}{2}/m$.

h the primary random hash function.

Claim

$\mathbf{E}\left[\sum_{i=0}^{m-1} Y_i^2\right] \leq 3n/2$ *if* $m \geq n$.

Proof.

Let C be total number of collisions. We already saw $\mathbf{E}[C] \leq \binom{n}{2}/m$.
$\sum_i \binom{Y_i}{2} = C$ and hence $\sum_i Y_i^2 = 2C + \sum_i Y_i$.
Therefore

$$\mathbf{E}\left[\sum_i Y_i^2\right] \leq 2\binom{n}{2}/m + \mathbf{E}\left[\sum_i Y_i\right] = 2\binom{n}{2}/m + n \leq 3n/2.$$

□

Perfect Hashing

- Do hashing once with table T of size m
- For each slot i in T let Y_i be number of elements hashed to slot i. If $Y_i > 1$ use perfect hashing with second table T_i of size Y_i^2.

Space usage is $Z = m + \sum_{i=0}^{m-1} Y_i^2$ and $\mathbf{E}[Z] \leq 5n/2$ if $m = n$.

- Use algorithm to create perfect hash table
- By Markov space usage is $< 5n$ with probability at least $1/2$
- Repeat if space usage is larger than $5n$. Expected number of repetitions is 2. Hence it leads to $O(n)$ time Las Vegas algorithm
- Technically also need to count the space to store multiple hash functions: $O(n)$ overhead

Rehashing, amortization and...

... making the hash table dynamic

So far we assumed fixed S of size $\simeq m$.

Question: What happens as items are inserted and deleted?

1. If $|S|$ grows to more than cm for some constant c then hash table performance clearly degrades.
2. If $|S|$ stays around $\simeq m$ but incurs many insertions and deletions then the initial random hash function is no longer random enough!

Rehashing, amortization and...

... making the hash table dynamic

So far we assumed fixed S of size $\simeq m$.

Question: What happens as items are inserted and deleted?

1. If $|S|$ grows to more than cm for some constant c then hash table performance clearly degrades.
2. If $|S|$ stays around $\simeq m$ but incurs many insertions and deletions then the initial random hash function is no longer random enough!

Solution: Rebuild hash table periodically!

1. Choose a new table size based on current number of elements in the table.
2. Choose a new random hash function and rehash the elements.
3. Discard old table and hash function.

Question: When to rebuild? How expensive?

Rebuilding the hash table

1. Start with table size m where m is some estimate of $|S|$ (can be some large constant).
2. If $|S|$ grows to more than twice current table size, build new hash table (choose a new random hash function) with double the current number of elements. Can also use similar trick if table size falls below quarter the size.

Rebuilding the hash table

1. Start with table size m where m is some estimate of $|S|$ (can be some large constant).
2. If $|S|$ grows to more than twice current table size, build new hash table (choose a new random hash function) with double the current number of elements. Can also use similar trick if table size falls below quarter the size.
3. If $|S|$ stays roughly the same but more than $c|S|$ operations on table for some chosen constant c (say $\mathbf{10}$), rebuild.

Rebuilding the hash table

1. Start with table size m where m is some estimate of $|S|$ (can be some large constant).
2. If $|S|$ grows to more than twice current table size, build new hash table (choose a new random hash function) with double the current number of elements. Can also use similar trick if table size falls below quarter the size.
3. If $|S|$ stays roughly the same but more than $c|S|$ operations on table for some chosen constant c (say $\mathbf{10}$), rebuild.

The **amortize** cost of rebuilding to previously performed operations. Rebuilding ensures $O(1)$ expected analysis holds even when S changes. Hence $O(1)$ expected look up/insert/delete time *dynamic* data dictionary data structure!

Practical Issues

Hashing used typically for integers, vectors, strings etc.

- Universal hashing is defined for integers. To implement for other objects need to map objects in some fashion to integers (via representation)
- Practical methods for various important cases such as vectors, strings are studied extensively. See http://en.wikipedia.org/wiki/Universal_hashing for some pointers.
- Details on Cuckoo hashing and its advantage over chaining http://en.wikipedia.org/wiki/Cuckoo_hashing.
- Relatively recent important paper bridging theory and practice of hashing. "The power of simple tabulation hashing" by Mikkel Thorup and Mihai Patrascu, 2011. See http://en.wikipedia.org/wiki/Tabulation_hashing

Part IV

Bloom Filters

Bloom Filters

Hashing:

1. To insert x in dictionary store x in table in location $h(x)$
2. To lookup y in dictionary check contents of location $h(y)$
3. Storing items in dictionary expensive in terms of memory, especially if items are unwieldy objects such a long strings, images, etc with *non-uniform* sizes.

Bloom Filters

Hashing:

1. To insert x in dictionary store x in table in location $h(x)$
2. To lookup y in dictionary check contents of location $h(y)$
3. Storing items in dictionary expensive in terms of memory, especially if items are unwieldy objects such a long strings, images, etc with *non-uniform* sizes.

Bloom Filter: tradeoff space for false positives

1. To insert x in dictionary set *bit* to $\mathbf{1}$ in location $h(x)$ (initially all bits are set to $\mathbf{0}$)
2. To lookup y if bit in location $h(y)$ is $\mathbf{1}$ say yes, else no.

Bloom Filters

Bloom Filter: tradeoff space for false positives

1. To insert x in dictionary set *bit* to $\mathbf{1}$ in location $h(x)$ (initially all bits are set to $\mathbf{0}$)
2. To lookup y if bit in location $h(y)$ is $\mathbf{1}$ say yes, else no
3. No false negatives but false positives possible due to collisions

Bloom Filters

Bloom Filter: tradeoff space for false positives

1. To insert x in dictionary set *bit* to $\mathbf{1}$ in location $h(x)$ (initially all bits are set to $\mathbf{0}$)
2. To lookup y if bit in location $h(y)$ is $\mathbf{1}$ say yes, else no
3. No false negatives but false positives possible due to collisions

Reducing false positives:

1. Pick k hash functions $h_1, h_2, \ldots, h_k$ *independently*

Bloom Filters

Bloom Filter: tradeoff space for false positives

1. To insert x in dictionary set *bit* to $\mathbf{1}$ in location $h(x)$ (initially all bits are set to $\mathbf{0}$)
2. To lookup y if bit in location $h(y)$ is $\mathbf{1}$ say yes, else no
3. No false negatives but false positives possible due to collisions

Reducing false positives:

1. Pick k hash functions $h_1, h_2, \ldots, h_k$ *independently*
2. To insert set $h_i(x)$th bit to one in table i for each $1 \leq i \leq k$

Bloom Filters

Bloom Filter: tradeoff space for false positives

1. To insert x in dictionary set *bit* to 1 in location $h(x)$ (initially all bits are set to 0)
2. To lookup y if bit in location $h(y)$ is 1 say yes, else no
3. No false negatives but false positives possible due to collisions

Reducing false positives:

1. Pick k hash functions $h_1, h_2, \ldots, h_k$ *independently*
2. To insert set $h_i(x)$th bit to one in table i for each $1 \leq i \leq k$
3. To lookup y compute $h_i(y)$ for $1 \leq i \leq k$ and say yes only if each bit in the corresponding location is 1, otherwise say no. If probability of false positive for one hash function is $\alpha < 1$ then with k independent hash function it is α^k.