

Limited independence and Hashing

Lecture 04

January 24, 2019

Pseudorandomness

Randomized algorithms rely on independent random bits

Pseudorandomness: when can we *avoid* or *limit* number of random bits?

- Motivated by fundamental theoretical questions and applications
- Applications: hashing, cryptography, streaming, simulations, derandomization, . . .
- A large topic in TCS with many connections to mathematics.

This course: need t -wise independent variables and hashing

Part I

t-wise independent random variables

Pairwise independent random variables

Definition

Random variables X_1, X_2, \dots, X_n from a range B are independent if for all $b_1, b_2, \dots, b_n \in B$

$$\Pr[X_1 = b_1, X_2 = b_2, \dots, X_n = b_n] = \prod_{i=1}^n \Pr[X_i = b_i].$$

Uniformly distributed if $\Pr[X_i = b] = 1/|B|$ for all $i, b \in B$.

Pairwise independent random variables

Definition

Random variables X_1, X_2, \dots, X_n from a range B are independent if for all $b_1, b_2, \dots, b_n \in B$

$$\Pr[X_1 = b_1, X_2 = b_2, \dots, X_n = b_n] = \prod_{i=1}^n \Pr[X_i = b_i].$$

Uniformly distributed if $\Pr[X_i = b] = 1/|B|$ for all $i, b \in B$.

Definition

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

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

Pairwise independent random variables

Definition

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

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

If X_1, X_2, \dots, X_n are independent then they are pairwise independent but converse is not necessarily true

Pairwise independent random variables

Definition

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

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

If X_1, X_2, \dots, X_n are independent then they are pairwise independent but converse is not necessarily true

Example: X_1, X_2 are independent bits (variables from $\{0, 1\}$) and $X_3 = X_1 \oplus X_2$. X_1, X_2, X_3 are pairwise independent but not independent.

Motivation for pairwise independence from streaming

Want n uniformly distr random variables X_1, X_2, \dots, X_n , say bits
But cannot store n bits because n is too large.

Achievable:

- storage of $O(\log n)$ random bits
- given i where $1 \leq i \leq n$ can generate X_i in $O(\log n)$ time
- X_1, X_2, \dots, X_n are pairwise independent and uniform
- Hence, with small storage, can generate n random variables “on the fly”. In several applications, pairwise independence (or generalizations) suffice

Generating pairwise independent bits

Assume for simplicity $n = 2^k - 1$ (otherwise consider nearest power of 2). Hence $k = O(\log n)$

- Let Y_1, Y_2, \dots, Y_k be independent bits
- For any $S \subset \{1, 2, \dots, k\}$, $S \neq \emptyset$, define $X_S = \bigoplus_{i \in S} Y_i$
- $2^k - 1$ random variables X_S

Generating pairwise independent bits

Assume for simplicity $n = 2^k - 1$ (otherwise consider nearest power of 2). Hence $k = O(\log n)$

- Let Y_1, Y_2, \dots, Y_k be independent bits
- For any $S \subset \{1, 2, \dots, k\}$, $S \neq \emptyset$, define $X_S = \bigoplus_{i \in S} Y_i$
- $2^k - 1$ random variables X_S

Claim: If $S \neq T$ then X_S and X_T are independent

Pairwise independent variables with larger range

Suppose we want n pairwise independent random variables in range $\{0, 1, 2, \dots, m - 1\}$

Pairwise independent variables with larger range

Suppose we want n pairwise independent random variables in range $\{0, 1, 2, \dots, m - 1\}$

- Now each X_i needs to be a $\log m$ bit string
- Use preceding construction for each bit independently
- Requires $O(\log m \log n)$ bits total
- Can in fact do $O(\log n + \log m)$ bits

Using prime numbers and fields

Assume $n = p$ and $m - 1 = p$ where p is a prime number

Want p pairwise random variables distributed uniformly in

$$\mathbb{Z}_p = \{0, 1, 2, \dots, p - 1\}$$

Using prime numbers and fields

Assume $n = p$ and $m - 1 = p$ where p is a prime number

Want p pairwise random variables distributed uniformly in $\mathbb{Z}_p = \{0, 1, 2, \dots, p - 1\}$

- Choose $a, b \in \{0, 1, 2, \dots, p - 1\}$ uniformly and independently at random. Requires $2\lceil \log p \rceil$ random bits
- For $0 \leq i \leq p - 1$ set $X_i = ai + b \pmod p$
- Note that one needs to store only a, b, p and can generate X_i efficiently on the fly

Using prime numbers and fields

Assume $n = p$ and $m - 1 = p$ where p is a prime number

Want p pairwise random variables distributed uniformly in $\mathbb{Z}_p = \{0, 1, 2, \dots, p - 1\}$

- Choose $a, b \in \{0, 1, 2, \dots, p - 1\}$ uniformly and independently at random. Requires $2\lceil \log p \rceil$ random bits
- For $0 \leq i \leq p - 1$ set $X_i = ai + b \pmod p$
- Note that one needs to store only a, b, p and can generate X_i efficiently on the fly

Exercise: Prove that each X_i is uniformly distributed in \mathbb{Z}_p .

Claim: For $i \neq j$, X_i and X_j are independent.

Using prime numbers and fields

Claim: For $i \neq j$, X_i and X_j are independent.

Some math required:

- \mathbb{Z}_p is a field for any prime p . That is $\{0, 1, 2, \dots, p - 1\}$ forms a commutative group under addition mod p (easy). And more importantly $\{1, 2, \dots, p - 1\}$ forms a commutative group under multiplication.

Some math required...

Lemma (LemmaUnique)

Let p be a prime number,

x : an integer number in $\{1, \dots, p - 1\}$.

\implies There exists a unique y s.t. $xy = 1 \pmod p$.

In other words: For every element there is a unique inverse.

$\implies \mathbb{Z}_p = \{0, 1, \dots, p - 1\}$ when working modulo p is a *field*.

Proof of Lemma Unique

Claim

Let p be a prime number. For any $x, y, z \in \{1, \dots, p-1\}$ s.t. $y \neq z$, we have that $xy \bmod p \neq xz \bmod p$.

Proof.

Assume for the sake of contradiction $xy \bmod p = xz \bmod p$.
Then

$$\begin{aligned}x(y - z) &= 0 \pmod{p} \\ \implies p &\text{ divides } x(y - z) \\ \implies p &\text{ divides } y - z \\ \implies y - z &= 0 \\ \implies y &= z.\end{aligned}$$

Proof of LemmaUnique

Lemma (LemmaUnique)

Let p be a prime number,

x : an integer number in $\{1, \dots, p - 1\}$.

\implies There exists a unique y s.t. $xy = 1 \pmod p$.

Proof.

By the above claim if $xy = 1 \pmod p$ and $xz = 1 \pmod p$ then $y = z$. Hence uniqueness follows.

Proof of LemmaUnique

Lemma (LemmaUnique)

Let p be a prime number,

x : an integer number in $\{1, \dots, p - 1\}$.

\implies There exists a unique y s.t. $xy = 1 \pmod p$.

Proof.

By the above claim if $xy = 1 \pmod p$ and $xz = 1 \pmod p$ then $y = z$. Hence uniqueness follows.

Existence. For any $x \in \{1, \dots, p - 1\}$ we have that $\{x * 1 \pmod p, x * 2 \pmod p, \dots, x * (p - 1) \pmod p\} = \{1, 2, \dots, p - 1\}$.

\implies There exists a number $y \in \{1, \dots, p - 1\}$ such that $xy = 1 \pmod p$.

Proof of pairwise independence

Lemma

If $x \neq y$ then for each

$(r, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$ there is exactly **one** pair $(a, b) \in \mathbb{Z}_p \times \mathbb{Z}_p$ such that
 $ax + b \pmod p = r$ and $ay + b \pmod p = s$.

Proof.

Solve the two equations:

$$ax + b = r \pmod p \quad \text{and} \quad ay + b = s \pmod p$$

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

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

Proof of pairwise independence

Lemma

If $x \neq y$ then for each

$(r, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$ there is exactly **one** pair $(a, b) \in \mathbb{Z}_p \times \mathbb{Z}_p$ such that
 $ax + b \pmod p = r$ and $ay + b \pmod p = s$.

Proof.

Solve the two equations:

$$ax + b = r \pmod p \quad \text{and} \quad ay + b = s \pmod p$$

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

One-to-one correspondence between (a, b) and (r, s)
 \Rightarrow if (a, b) is uniformly at random from \mathbb{Z}_p then (r, s) is uniformly at random from \mathbb{Z}_p

Pairwise Independence and Chebyshev's Inequality

Chebyshev's Inequality

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

Suppose $X = X_1 + X_2 + \dots + X_n$.

If X_1, X_2, \dots, X_n are independent then $\text{Var}(X) = \sum_i \text{Var}(X_i)$.

Recall application to random walk on line

Pairwise Independence and Chebyshev's Inequality

Chebyshev's Inequality

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

Suppose $X = X_1 + X_2 + \dots + X_n$.

If X_1, X_2, \dots, X_n are independent then $\text{Var}(X) = \sum_i \text{Var}(X_i)$.

Recall application to random walk on line

Lemma

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

Pairwise independence for arbitrary 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 for arbitrary n and m

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 (we only used that \mathbb{Z}_p is a field) X_i 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 useful here).

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

t -wise independence

Generalizing pairwise independence:

Definition

Random variables X_1, X_2, \dots, X_n from a range B are t -wise independent for integer $t > 1$ if $X_{i_1}, X_{i_2}, \dots, X_{i_t}$ are independent for any $i_1 \neq i_2 \neq \dots \neq i_t \in \{1, 2, \dots, n\}$.

As t increases the variables become more and more independent. If $t = n$ the variables are independent.

t -wise independence

Generalizing pairwise independence:

Definition

Random variables X_1, X_2, \dots, X_n from a range B are t -wise independent for integer $t > 1$ if $X_{i_1}, X_{i_2}, \dots, X_{i_t}$ are independent for any $i_1 \neq i_2 \neq \dots \neq i_t \in \{1, 2, \dots, n\}$.

As t increases the variables become more and more independent. If $t = n$ the variables are independent.

Fact: For any n, m one can create n random t -wise independent random variables from the range $[m]$ using $O(t(\log n + \log m))$ true random bits. Can store only bits and generate the variables on the fly in $O(t \text{polylog}(m + n))$ time.

t -wise independence

Construction using polynomials

- Let \mathbb{F} be a field
- Pick t random (with replacement) numbers from \mathbb{F} :
 a_0, a_1, \dots, a_{t-1}
- For each $i \in [|\mathbb{F}|]$ set $X_i = a_0 + a_1i + a_2i^2 + \dots + a_{t-1}i^{t-1}$

Part II

Hashing

Balls and Bins and Load Balancing

Suppose we want to distribute jobs to machines in a simple way to achieve load balancing.

Throwing each new job into a random machine is a simple, distributed, oblivious strategy with many benefits

Balls and bins is simple mathematical model to analyze the core principles

Balls and Bins \rightarrow Hashing

Hashing:

- Want a “function” $h : \mathcal{U} \rightarrow \mathcal{B}$.
- Want h to behave like a “random function”. That is for any distinct $x_1, x_2, \dots, x_n \in \mathcal{U}$ we have $h(x_1), h(x_2), \dots, h(x_n)$ to be uniformly distributed over \mathcal{B} and independent.
- But want h to be efficiently computable and stored in small memory

Balls and Bins \rightarrow Hashing

Hashing:

- Want a “function” $h : \mathcal{U} \rightarrow \mathcal{B}$.
- Want h to behave like a “random function”. That is for any distinct $x_1, x_2, \dots, x_n \in \mathcal{U}$ we have $h(x_1), h(x_2), \dots, h(x_n)$ to be uniformly distributed over \mathcal{B} and independent.
- But want h to be efficiently computable and stored in small memory

Many applications: hash tables as dictionary data structure, cryptography/security, pseudorandomness, ...

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 $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$.
- 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 $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$.
- 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?

Hashing and Hash Tables

Hash Table data structure:

- 1 A (hash) table/array T of size m (the table **size**).
- 2 A hash function $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$.
- 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!

Hashing and Hash Tables

Hash Table data structure:

- 1 A (hash) table/array T of size m (the table **size**).
- 2 A hash function $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$.
- 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!

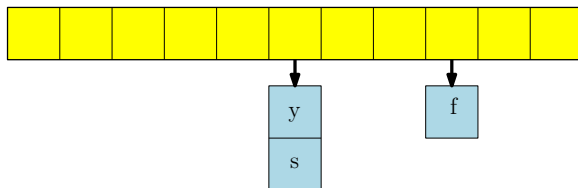
Collisions unavoidable if $|T| < |\mathcal{U}|$. Several techniques to handle them.

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.



Chain length determines time for operations. Ideally want $O(1)$.

Hash Functions

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

Goal: $O(1)$ -time lookup, insertion, deletion.

Single hash function

If $N \geq m^2$, then for any hash function $h: \mathcal{U} \rightarrow \mathcal{T}$ there exists $i < m$ such that at least $N/m \geq m$ elements of \mathcal{U} get hashed to slot i .

Hash Functions

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

Goal: $O(1)$ -time lookup, insertion, deletion.

Single hash function

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

Hash Functions

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

Goal: $O(1)$ -time lookup, insertion, deletion.

Single hash function

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

Hash Functions

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

Goal: $O(1)$ -time lookup, insertion, deletion.

Single hash function

If $N \geq m^2$, then for any hash function $h: \mathcal{U} \rightarrow \mathcal{T}$ there exists $i < m$ such that at least $N/m \geq m$ elements of \mathcal{U} get hashed to slot i . Any \mathcal{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

Question: What are good properties of \mathcal{H} in distributing data?

Strongly Universal Hashing

Question: What are good properties of \mathcal{H} in distributing data?

- 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 \mathcal{T} . In other words $\Pr[h(x) = i] = 1/m$ for every $0 \leq i < m$.

Strongly Universal Hashing

Question: What are good properties of \mathcal{H} in distributing data?

- 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 \mathcal{T} . In other words $\Pr[h(x) = i] = 1/m$ for every $0 \leq i < m$.
- 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

Question: What are good properties of \mathcal{H} in distributing data?

- **(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.

(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.

(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.

Generalizes to t -strongly universal and t -universal families. Need property for any tuple of t items.

Analyzing Universal Hashing

Question: Fixing set S , what is the *expected* time to look up $x \in S$ when h is picked uniformly at random from \mathcal{H} ?

- 1 $\ell(x)$: the size of the list at $T[h(x)]$. We want $E[\ell(x)]$
- 2 For $y \in S$ let D_y be one if $h(y) = h(x)$, else zero.
$$\ell(x) = \sum_{y \in S} D_y$$

Analyzing Universal Hashing

Question: Fixing set S , what is the *expected* time to look up $x \in S$ when h is picked uniformly at random from \mathcal{H} ?

① $\ell(x)$: the size of the list at $T[h(x)]$. We want $E[\ell(x)]$

② For $y \in S$ let D_y be one if $h(y) = h(x)$, else zero.

$$\ell(x) = \sum_{y \in S} D_y$$

$$\begin{aligned} E[\ell(x)] &= \sum_{y \in S} E[D_y] = \sum_{y \in S} \Pr[h(x) = h(y)] \\ &= \sum_{y \in S} \frac{1}{m} \quad (\text{since } \mathcal{H} \text{ is a universal hash family}) \\ &= |S|/m \leq 1 \quad \text{if } |S| \leq m \end{aligned}$$

Analyzing Universal Hashing

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

Answer: $O(n/m)$.

Analyzing Universal Hashing

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

Answer: $O(n/m)$.

Comments:

- 1 $O(1)$ expected time also holds for insertion.
- 2 Analysis assumes static set S but holds as long as S is a set formed with at most $O(m)$ insertions and deletions.
- 3 **Worst-case:** look up time can be large! How large? In principle $\Omega(n)$ time but if \mathcal{H} has good properties then $O(\sqrt{n})$ or $O(\log n / \log \log n)$ with high probability.

Universal Hash Family

Universal: \mathcal{H} such that $\Pr[h(x) = h(y)] = 1/m$.

All functions

\mathcal{H} : Set of all possible functions $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$.

- Universal.

Universal Hash Family

Universal: \mathcal{H} such that $\Pr[h(x) = h(y)] = 1/m$.

All functions

\mathcal{H} : Set of all possible functions $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$.

- Universal.
- $|\mathcal{H}| = m^{|\mathcal{U}|}$
- representing h requires $|\mathcal{U}| \log m$ – Not $O(1)$!

Universal Hash Family

Universal: \mathcal{H} such that $\Pr[h(x) = h(y)] = 1/m$.

All functions

\mathcal{H} : Set of all possible functions $h : \mathcal{U} \rightarrow \{0, \dots, m - 1\}$.

- Universal.
- $|\mathcal{H}| = m^{|\mathcal{U}|}$
- representing h requires $|\mathcal{U}| \log m$ – Not $O(1)$!

We need *compactly representable* universal family.

Compact Stongly Universal Hash Family

Similar to construction of N pairwise independent random variables with range $[m]$.

The function is given by the algorithm to construct X_i given i .

Can do with $O(\log N)$ bits of storage since $N \geq m$ in hashing application.

A Compact Universal Hash Family

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

- 1 Choose a **prime** number $p \geq N$. $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$ is a field.
- 2 For $a, b \in \mathbb{Z}_p$, $a \neq 0$, define the hash function $h_{a,b}$ as $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$.
- 3 Let $\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$. Note that $|\mathcal{H}| = p(p-1)$.

A Compact Universal Hash Family

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

- 1 Choose a **prime** number $p \geq N$. $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$ is a field.
- 2 For $a, b \in \mathbb{Z}_p$, $a \neq 0$, define the hash function $h_{a,b}$ as $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$.
- 3 Let $\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$. Note that $|\mathcal{H}| = p(p-1)$.

Theorem

\mathcal{H} is a universal hash family.

A Compact Universal Hash Family

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

- 1 Choose a **prime** number $p \geq N$. $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$ is a field.
- 2 For $a, b \in \mathbb{Z}_p$, $a \neq 0$, define the hash function $h_{a,b}$ as $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$.
- 3 Let $\mathcal{H} = \{h_{a,b} \mid a, b \in \mathbb{Z}_p, a \neq 0\}$. Note that $|\mathcal{H}| = p(p-1)$.

Theorem

\mathcal{H} is a universal hash family.

Comments:

- 1 Hash family is of small size, easy to sample from.
- 2 Easy to store a hash function (a, b have to be stored) and evaluate it.

A Compact Universal Hash Family

- $g(x) = ax + b$ is uniformly distributed in $\{0, 1, \dots, p - 1\}$ but $h(x)$ is not uniformly distributed unless $m = p$.
- $\Pr[h(x) = i] \leq 2/m$ for any i .

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 $h(y)$

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 $h(y)$

Bloom Filter: tradeoff space for false positives

- 1 Storing items in dictionary expensive in terms of memory, especially if items are unwieldy objects such as long strings, images, etc with *non-uniform* sizes.
- 2 To insert x in dictionary set *bit* to **1** in location $h(x)$ (initially all bits are set to **0**)
- 3 To lookup y if bit in location $h(y)$ is **1** say yes, else no.

Bloom Filters

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, \dots, h_k *independently*
- 2 To insert x , for each i , set bit in location $h_i(x)$ in table i to **1**
- 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 .

Take away points

- ① Hashing is a powerful and important technique for dictionaries. Many practical applications.
- ② Randomization fundamental to understanding hashing.
- ③ Good and efficient hashing possible in theory and practice with proper definitions (universal, perfect, etc).
- ④ Related ideas of creating a compact fingerprint/sketch for objects is very powerful in theory and practice.

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.
- 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