Lecture 10: Universal and Perfect Hashing

Michael Dinitz

September 26, 2024
601.433/633 Introduction to Algorithms

Introduction

Another approach to dictionaries (insert, lookup, delete): hashing

- Can improve operations to $O(1)$, but with many caveats!

Should have seen some discussion of hashing in data structures. Also in CLRS.

- Separate chaining vs. open addressing

Today: discussion of caveats, more advanced versions of hashing (universal and perfect)

Hashing Basics

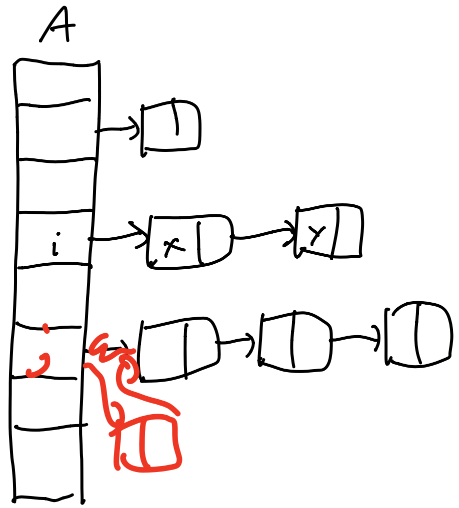
- Keys from universe $\boldsymbol{U}$ (think very large)
- Set $\boldsymbol{S} \subseteq \boldsymbol{U}$ of keys we actually care about (think relatively small). $|\boldsymbol{S}| = \boldsymbol{N}$.
- Hash table $\boldsymbol{A}$ (array) of size $\boldsymbol{M}$.
- Hash function $\boldsymbol{h} : \boldsymbol{U} \to [\boldsymbol{M}]$
  - $[\boldsymbol{M}] = \{1, 2, \ldots, \boldsymbol{M}\}$
- Idea: store $\boldsymbol{x}$ in $\boldsymbol{A}[\boldsymbol{h}(\boldsymbol{x})]$

Hashing Basics

- Keys from universe U (think very large)
- Set $S \subseteq U$ of keys we actually care about (think relatively small). $|S| = N$.
- Hash table A (array) of size M.
- Hash function $h : U \to [M]$
  - $[M] = \{1, 2, \ldots, M\}$
- Idea: store x in $A[h(x)]$

One more component: *collision resolution*

- Today: *separate chaining*
- $A[i]$ is a linked list containing all x inserted where $h(x) = i$.



Dictionary Operations

Lookup($\boldsymbol{x}$): Walk down the list at $\boldsymbol{A}[\boldsymbol{h}(\boldsymbol{x})]$ until we find $\boldsymbol{x}$ (or walk to the end of the list)

Insert($\boldsymbol{x}$): Add $\boldsymbol{x}$ to the beginning of the list at $\boldsymbol{A}[\boldsymbol{h}(\boldsymbol{x})]$.

Delete($\boldsymbol{x}$): Walk down the list at $\boldsymbol{A}[\boldsymbol{h}(\boldsymbol{x})]$ until we find $\boldsymbol{x}$. Remove it from the list.

Dictionary Operations

Lookup(x): Walk down the list at $A[h(x)]$ until we find x (or walk to the end of the list)

Insert(x): Add x to the beginning of the list at $A[h(x)]$.

Delete(x): Walk down the list at $A[h(x)]$ until we find x. Remove it from the list.

Question: What should hash function be?

Dictionary Operations

Lookup(x): Walk down the list at $A[h(x)]$ until we find x (or walk to the end of the list)

Insert(x): Add x to the beginning of the list at $A[h(x)]$.

Delete(x): Walk down the list at $A[h(x)]$ until we find x. Remove it from the list.

Question: What should hash function be?

Properties we want:

Dictionary Operations

Lookup(x): Walk down the list at $A[h(x)]$ until we find x (or walk to the end of the list)

Insert(x): Add x to the beginning of the list at $A[h(x)]$.

Delete(x): Walk down the list at $A[h(x)]$ until we find x. Remove it from the list.

Question: What should hash function be?

Properties we want:

- Few collisions. Time of lookup, delete for x is O(length of list at $A[h(x)]$).

Dictionary Operations

Lookup(x): Walk down the list at $A[h(x)]$ until we find x (or walk to the end of the list)

Insert(x): Add x to the beginning of the list at $A[h(x)]$.

Delete(x): Walk down the list at $A[h(x)]$ until we find x. Remove it from the list.

Question: What should hash function be?

Properties we want:

- Few collisions. Time of lookup, delete for x is O(length of list at $A[h(x)]$).
- Small M. Ideally, $M = O(N)$.

Dictionary Operations

Lookup(x): Walk down the list at $A[h(x)]$ until we find x (or walk to the end of the list)

Insert(x): Add x to the beginning of the list at $A[h(x)]$.

Delete(x): Walk down the list at $A[h(x)]$ until we find x. Remove it from the list.

Question: What should hash function be?

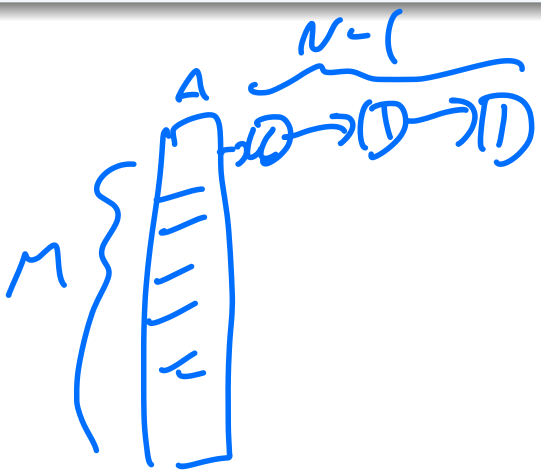
Properties we want:

- Few collisions. Time of lookup, delete for x is O(length of list at $A[h(x)]$).
- Small M. Ideally, $M = O(N)$.
- h fast to compute.

Bad News

Theorem

For any hash function h, if $|U| \geq (N-1)M + 1$, then there exists a set S of N elements that all hash to the same location.



Bad News

Theorem

For any hash function h, if $|U| \geq (N-1)M+1$, then there exists a set S of N elements that all hash to the same location.

Proof.

Pigeonhole principle / contradiction / contrapositive. □

Bad News

Theorem

For any hash function h, if $|U| \geq (N-1)M+1$, then there exists a set S of N elements that all hash to the same location.

Proof.

Pigeonhole principle / contradiction / contrapositive. □

So worst case behavior always bad! How can we get around this?

Bad News

Theorem

For any hash function $\boldsymbol{h}$*, if* $|\boldsymbol{U}| \geq (\boldsymbol{N}-1)\boldsymbol{M}+1$*, then there exists a set* $\boldsymbol{S}$ *of* $\boldsymbol{N}$ *elements that all hash to the same location.*

Proof.

Pigeonhole principle / contradiction / contrapositive. □

So worst case behavior always bad! How can we get around this?

- Option 1: don't worry about it, hope world isn't adversarial.

Bad News

Theorem

For any hash function h, if $|U| \geq (N-1)M+1$, then there exists a set S of N elements that all hash to the same location.

Proof.

Pigeonhole principle / contradiction / contrapositive. $\square$

So worst case behavior always bad! How can we get around this?

- Option 1: don't worry about it, hope world isn't adversarial.
- Option 2: Randomness! *Random function* $h : U \to [M]$
  - For each $x \in U$, choose $y \in [M]$ uniformly at random and set $h(x) = y$.
  - Hopefully good behavior in expectation.

Bad News

Theorem

For any hash function h, if $|U| \geq (N-1)M + 1$, then there exists a set S of N elements that all hash to the same location.

Proof.

Pigeonhole principle / contradiction / contrapositive. □

So worst case behavior always bad! How can we get around this?

- Option 1: don't worry about it, hope world isn't adversarial.
- Option 2: Randomness! *Random function* $h : U \to [M]$
  - For each $x \in U$, choose $y \in [M]$ uniformly at random and set $h(x) = y$.
  - Hopefully good behavior in expectation.
  - Problem: How can we store/remember/create h?

Universal Hashing

Definition

A probability distribution H over hash functions $\{h : U \to [M]\}$ is *universal* if

$$\Pr_{h \sim H}[h(x) = h(y)] \le 1/M$$

for all $x, y \in U$ with $x \neq y$.

Universal Hashing

Definition

A probability distribution H over hash functions $\{h : U \to [M]\}$ is *universal* if

$$\Pr_{h \sim H}[h(x) = h(y)] \leq 1/M$$

for all $x, y \in U$ with $x \neq y$.

Clearly satisfied by H = uniform distribution over all hash functions

Universal Hashing

Definition

A probability distribution H over hash functions $\{h : U \to [M]\}$ is *universal* if

$$\Pr_{h \sim H}[h(x) = h(y)] \le 1/M$$

for all $x, y \in U$ with $x \neq y$.

Clearly satisfied by H = uniform distribution over all hash functions

Theorem

If H is universal, then for every set $S \subseteq U$ with $|S| = N$ and for every $x \in U$, the expected number of collisions (when we draw h from H) between x and elements of S is at most N/M.

Universal Hashing

Definition

A probability distribution H over hash functions $\{h : U \to [M]\}$ is *universal* if

$$\Pr_{h \sim H}[h(x) = h(y)] \leq 1/M$$

for all $x, y \in U$ with $x \neq y$.

Clearly satisfied by H = uniform distribution over all hash functions

Theorem

If H is universal, then for every set $S \subseteq U$ with $|S| = N$ and for every $x \in U$, the expected number of collisions (when we draw h from H) between x and elements of S is at most N/M.

So Lookup(x) and Delete(x) have expected time $O(N/M)$.
$\implies$ If $M = \Omega(N)$, operations in $O(1)$ time!

Main Proof

Theorem

If H is universal, then for every set $S \subseteq U$ with $|S| = N$ and for every $x \in U$, the expected number of collisions (when we draw h from H) between x and elements of S is at most N/M.

Proof.

Let $C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$

$$\Longrightarrow E[C_{xy}] = \Pr_{h \sim H}[h(x) = h(y)] \leq 1/M$$

Main Proof

Theorem

If H is universal, then for every set $S \subseteq U$ with $|S| = N$ and for every $x \in U$, the expected number of collisions (when we draw h from H) between x and elements of S is at most N/M.

Proof.

Let $C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$

$$\implies E[C_{xy}] = \Pr_{h \sim H}[h(x) = h(y)] \leq 1/M$$

Number of collisions between x and S is exactly $\sum_{y \in S} C_{xy}$

$$\implies E\left[\sum_{y \in S} C_{xy}\right] = \sum_{y \in S} E\left[C_{xy}\right] \leq \sum_{y \in S} \frac{1}{M} = N/M$$

□

Main Corollary

Corollary

If H is universal, then for any sequence of L insert, lookup, and delete operations in which there are at most $O(M)$ elements in the system at any time, the expected total cost of the whole sequence is only $O(L)$ (assuming h takes constant time to compute).

Main Corollary

Corollary

If H is universal, then for any sequence of L insert, lookup, and delete operations in which there are at most $O(M)$ elements in the system at any time, the expected total cost of the whole sequence is only $O(L)$ (assuming h takes constant time to compute).

Proof.

By theorem, each operation $O(1)$ in expectation. Total time is sum: linearity of expectations. □

Main Corollary

Corollary

If H is universal, then for any sequence of L insert, lookup, and delete operations in which there are at most $O(M)$ elements in the system at any time, the expected total cost of the whole sequence is only $O(L)$ (assuming h takes constant time to compute).

Proof.

By theorem, each operation $O(1)$ in expectation. Total time is sum: linearity of expectations. □

So universal distributions are great. Can we construct them?

Universal Hash Families

Definition

If $\boldsymbol{H}$ is universal and is a uniform distribution over a set of functions $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots\}$, then that set is called a *universal hash family*.

Often use $\boldsymbol{H}$ to refer to both set of functions and uniform distribution over it.

Universal Hash Families

Definition

If H is universal and is a uniform distribution over a set of functions $\{h_1, h_2, \ldots\}$, then that set is called a *universal hash family*.

Often use H to refer to both set of functions and uniform distribution over it.

Notation:

- $U = \{0, 1\}^u$ (so $|U| = 2^u$)
- $M = 2^b$, so an index to A is an element of $\{0, 1\}^b$

Universal Hash Families

Definition

If H is universal and is a uniform distribution over a set of functions $\{h_1, h_2, \ldots\}$, then that set is called a *universal hash family*.

Often use H to refer to both set of functions and uniform distribution over it.

Notation:

- $U = \{0,1\}^u$ (so $|U| = 2^u$)
- $M = 2^b$, so an index to A is an element of $\{0,1\}^b$

Construction: $H = \{0,1\}^{b \times u}$, i.e., H is all $b \times u$ binary matrices

- Each $h \in H$ is a (linear) function from U to $[M]$:
  $h(x) = hx \in \{0,1\}^b$ (all operations mod 2)

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}}_{h} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}}_{h(x)}$$

Universality

Theorem

H is a universal hash family: $\mathbf{Pr}_{h \sim H}[h(x) = h(y)] \leq 1/M$ for all $x \neq y \in \{0,1\}^u$.

Universality

Theorem

H is a universal hash family: $\mathbf{Pr}_{h \sim H}[h(x) = h(y)] \leq 1/M$ for all $x \neq y \in \{0,1\}^u$.

Proof.

Matrix multiplication: $h(x) = hx = \sum_{i : x_i = 1} h^i$ (where h^i is i'th column of h).

Universality

Theorem

H is a universal hash family: $\mathbf{Pr}_{h \sim H}[h(x) = h(y)] \leq 1/M$ *for all* $x \neq y \in \{0,1\}^u$.

Proof.

Matrix multiplication: $h(x) = hx = \sum_{i : x_i = 1} h^i$ (where h^i is i'th column of h).

Since $x \neq y$, there is i s.t. $x_i \neq y_i$. WLOG, $x_i = 0$ and $y_i = 1$.

Universality

Theorem

***H** is a universal hash family:* $\mathbf{Pr}_{h \sim H}[h(x) = h(y)] \leq 1/M$ *for all* $x \neq y \in \{0,1\}^u$.

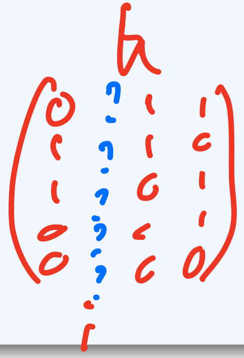
Proof.

Matrix multiplication: $h(x) = hx = \sum_{i:x_i=1} h^i$ (where h^i is i'th column of h).

Since $x \neq y$, there is i s.t. $x_i \neq y_i$. WLOG, $x_i = 0$ and $y_i = 1$.

Draw all entries of h *except* for h^i. Let $h' = h$ with h^i all 0's

- $h(x) = h'(x)$ already fixed.



Universality

Theorem

$\boldsymbol{H}$ *is a universal hash family:* $\mathbf{Pr}_{h\sim H}[h(x) = h(y)] \leq 1/M$ *for all* $x \neq y \in \{0,1\}^u$.

Proof.

Matrix multiplication: $h(x) = hx = \sum_{i:x_i=1} h^i$ (where h^i is i'th column of h).

Since $x \neq y$, there is i s.t. $x_i \neq y_i$. WLOG, $x_i = 0$ and $y_i = 1$.

Draw all entries of h *except* for h^i. Let $h' = h$ with h^i all 0's

- $h(x) = h'(x)$ already fixed.
- If $h(y) = h(x)$, then h^i must equal $h(x) - h'(y)$

Universality

Theorem

H is a universal hash family: $\mathbf{Pr}_{h\sim H}[h(x)=h(y)] \le 1/M$ for all $x \neq y \in \{0,1\}^u$.

Proof.

Matrix multiplication: $h(x) = hx = \sum_{i:x_i=1} h^i$ (where h^i is i'th column of h).

Since $x \neq y$, there is i s.t. $x_i \neq y_i$. WLOG, $x_i = 0$ and $y_i = 1$.

Draw all entries of h *except* for h^i. Let $h' = h$ with h^i all 0's

- $h(x) = h'(x)$ already fixed.
- If $h(y) = h(x)$, then h^i must equal $h(x) - h'(y)$
- Happens with probability exactly $1/2^b = 1/M$ □

Perfect Hashing

Suppose you know $\boldsymbol{S}$, never changes.

- Build table, then do lookups. Like a real dictionary!
- Care more about time to do lookup than time to build dictionary

Perfect Hashing

Suppose you know $\boldsymbol{S}$, never changes.

- Build table, then do lookups. Like a real dictionary!
- Care more about time to do lookup than time to build dictionary

Obvious approaches:

- Sorted array: lookups $\boldsymbol{O(\log N)}$
- Balanced search tree: $\boldsymbol{O(\log N)}$

Perfect Hashing

Suppose you know $\boldsymbol{S}$, never changes.

- Build table, then do lookups. Like a real dictionary!
- Care more about time to do lookup than time to build dictionary

Obvious approaches:

- Sorted array: lookups $\boldsymbol{O(\log N)}$
- Balanced search tree: $\boldsymbol{O(\log N)}$

Can we do better with hashing?

Perfect Hashing

Suppose you know S, never changes.

- Build table, then do lookups. Like a real dictionary!
- Care more about time to do lookup than time to build dictionary

Obvious approaches:

- Sorted array: lookups $O(\log N)$
- Balanced search tree: $O(\log N)$

Can we do better with hashing? Yes, through universal hashing!

Method 1

Use table of size $M = N^2$.

Method 1

Use table of size $M = N^2$.

Theorem

Let H be universal with $M = N^2$. Then $\mathbf{Pr}_{h \sim H}[\text{no collisions in } S] \geq 1/2$.

Proof.

Fix $x, y \in S$ with $x \neq y$.

Method 1

Use table of size $M = N^2$.

Theorem

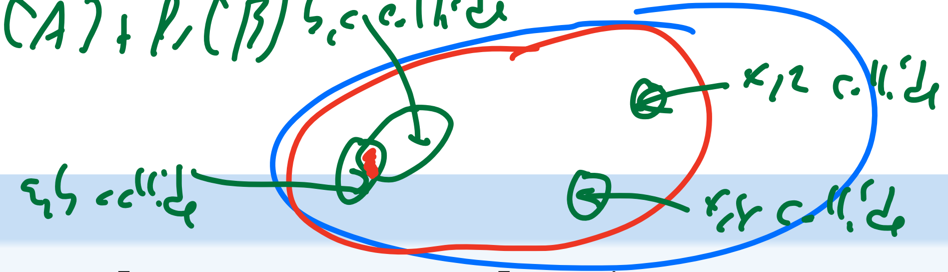
Let H be universal with $M = N^2$. Then $\Pr_{h \sim H}[\text{no collisions in } S] \geq 1/2$.

Proof.

Fix $x, y \in S$ with $x \neq y$.
$\Pr_{h \sim H}[h(x) = h(y)] \leq 1/M = 1/N^2$ by universality.

Method 1

Use table of size $M = N^2$.



Theorem

Let H be universal with $M = N^2$. Then $\mathbf{Pr}_{h\sim H}[$no collisions in $S] \geq 1/2$.

Proof.

Fix $x, y \in S$ with $x \neq y$.

$\mathbf{Pr}_{h\sim H}[h(x) = h(y)] \leq 1/M = 1/N^2$ by universality.

$$\mathbf{Pr}_{h\sim H}[\exists \text{ collision in } S] \leq \sum_{\substack{x,y\in S \\ x\neq y}} \mathbf{Pr}_{h\sim H}[h(x) = h(y)] \leq \sum_{\substack{x,y\in S \\ x\neq y}} \frac{1}{N^2}$$

$$= \binom{N}{2}\frac{1}{N^2} = \frac{N(N-1)}{2}\frac{1}{N^2} \leq \frac{1}{2}$$

□

Method 1

Use table of size $M = N^2$.

Theorem

Let H be universal with $M = N^2$. Then $\Pr_{h\sim H}[\text{no collisions in } S] \geq 1/2$.

Proof.

Fix $x, y \in S$ with $x \neq y$.

$\Pr_{h\sim H}[h(x) = h(y)] \leq 1/M = 1/N^2$ by universality.

$$\Pr_{h\sim H}[\exists \text{ collision in } S] \leq \sum_{\substack{x,y\in S\\ x\neq y}} \Pr_{h\sim H}[h(x) = h(y)] \leq \sum_{\substack{x,y\in S\\ x\neq y}} \frac{1}{N^2}$$

$$= \binom{N}{2}\frac{1}{N^2} = \frac{N(N-1)}{2}\frac{1}{N^2} \leq \frac{1}{2}$$

□

So keep sampling $h \sim H$ until get one with no collisions!

Method 2

$M = N^2$ is pretty big!

- Only storing N things, and know them ahead of time
- Want space $O(N)$
- Open question for a long time!

Method 2

$M = N^2$ is pretty big!

- Only storing N things, and know them ahead of time
- Want space $O(N)$
- Open question for a long time!

Starting approach: set $M = N$, use a universal hash family H. Draw $h \sim H$.

- Will have collisions. Need to do something other than chaining.

Method 2

$M = N^2$ is pretty big!

- Only storing N things, and know them ahead of time
- Want space $O(N)$
- Open question for a long time!

Starting approach: set $M = N$, use a universal hash family H. Draw $h \sim H$.

- Will have collisions. Need to do something other than chaining.

For each $i \in [M]$, let $S_i = \{x \in S : h(x) = i\}$ and let $n_i = |S_i|$

Method 2

$M = N^2$ is pretty big!

- Only storing N things, and know them ahead of time
- Want space $O(N)$
- Open question for a long time!

Starting approach: set $M = N$, use a universal hash family H. Draw $h \sim H$.

- Will have collisions. Need to do something other than chaining.

For each $i \in [M]$, let $S_i = \{x \in S : h(x) = i\}$ and let $n_i = |S_i|$

- Use another hash table for S_i!
- Use Method 1: $O(n_i^2)$-size perfect hashing of S_i.
  - Let $h_i : U \to [n_i^2]$ be hash function for S_i, and A_i be table (pointer from $A[i]$)

Method 2

$M = N^2$ is pretty big!

- Only storing N things, and know them ahead of time
- Want space $O(N)$
- Open question for a long time!

Starting approach: set $M = N$, use a universal hash family H. Draw $h \sim H$.

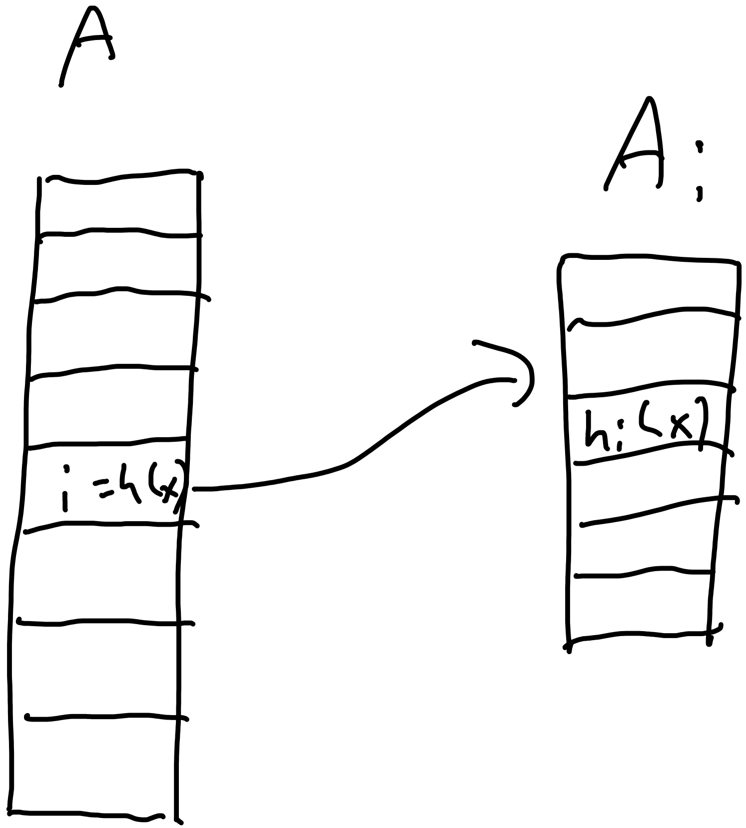
- Will have collisions. Need to do something other than chaining.

For each $i \in [M]$, let $S_i = \{x \in S : h(x) = i\}$ and let $n_i = |S_i|$

- Use another hash table for S_i!
- Use Method 1: $O(n_i^2)$-size perfect hashing of S_i.
  - Let $h_i : U \to [n_i^2]$ be hash function for S_i, and A_i be table (pointer from $A[i]$)

Lookup(x): Look in $A_{h(x)}[h_{h(x)}(x)]$

Picture



Analysis

Lookup time: by analysis of Method 1, no collisions in second level.

$\implies$ Lookup time $\boldsymbol{O(1)}$

Analysis

Lookup time: by analysis of Method 1, no collisions in second level.

$\Longrightarrow$ Lookup time $O(1)$

Size: $O(N + \sum_{i=1}^{N} n_i^2)$

Analysis

Lookup time: by analysis of Method 1, no collisions in second level.

$\implies$ Lookup time $O(1)$

Size: $O(N + \sum_{i=1}^{N} n_i^2)$

Theorem

Let H be universal onto a table of size N. Then

$$\Pr_{h \sim H}\left[\sum_{i=1}^{N} n_i^2 > 4N\right] < 1/2.$$

So like with method 1: keep drawing $h \sim H$ until $\sum_{i=1}^{N} n_i^2 \leq 4N$

Analysis

Lookup time: by analysis of Method 1, no collisions in second level.

$\Longrightarrow$ Lookup time $O(1)$

Size: $O(N + \sum_{i=1}^{N} n_i^2)$

Theorem

Let H be universal onto a table of size N. Then

$$\Pr_{h \sim H}\left[\sum_{i=1}^{N} n_i^2 > 4N\right] < 1/2.$$

So like with method 1: keep drawing $h \sim H$ until $\sum_{i=1}^{N} n_i^2 \leq 4N$

Prove that $E\left[\sum_{i=1}^{N} n_i^2\right] \leq 2N$.

- Implies theorem by Markov's inequality
  - $\Pr[X > 2E[X]] \leq 1/2$ for nonnegative random variables X.

Proof

Observation: $\sum_{i=1}^{N} n_i^2$ is exactly number of *ordered* pairs that collide, including self-collisions

- Example: If $S_i = \{a, b, c\}$ then $n_i^2 = 9$. Ordered colliding pairs: $(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)$

Proof

Observation: $\sum_{i=1}^{N} n_i^2$ is exactly number of *ordered* pairs that collide, including self-collisions

- Example: If $S_i = \{a, b, c\}$ then $n_i^2 = 9$. Ordered colliding pairs: $(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c,a),(c,b),(c,c)$

Let $C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$

Proof

Observation: $\sum_{i=1}^{N} n_i^2$ is exactly number of *ordered* pairs that collide, including self-collisions

- Example: If $S_i = \{a, b, c\}$ then $n_i^2 = 9$. Ordered colliding pairs: $(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c,a),(c,b),(c,c)$

Let $C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$

$$
\begin{aligned}
E\left[\sum_{i=1}^{N} n_i^2\right] &= E\left[\sum_{x \in S}\sum_{y \in S} C_{xy}\right] \\
&= N + \sum_{x \in S}\sum_{y \in S : y \neq x} E\left[C_{xy}\right] && \text{(linearity of expectations)} \\
&\le N + \frac{N(N-1)}{M} && \text{(definition of universal)} \\
&< 2N && \text{(since } M = N\text{)}
\end{aligned}
$$