ساختمان داده و الگوریتم ها (CE203)

جلسه هجدهم:
درهم سازی تصادفی

سجاد شیرعلی شهرضا

پاییز 1400

دوشنبه، 15 آذر 1400

Based on Stanford CS161 Slides from Summer 2020 by Karey Shi

اطلاع رسانی

- بخش مرتبط کتاب برای این جلسه: 11
- قرار دادن تمرین سوم بر روی سایت درس
  - مهلت ارسال: روز چهارشنبه 24 آذر خرداد 1400

تصادفی کردن تابع درهم ساز

چگونه میتوان بدخواهان را تضعیف کرد؟

INTUITION

Intuitively, the adversary can't foil a hash function that they don't yet know.

So, our strategy is to define a set of hash functions, and then we randomly choose a hash function **h** from this set to use!

INTUITION

Intuitively, the adversary can't foil a hash function that they don't yet know.

So, our strategy is to define a set of hash functions, and then we randomly choose a hash function **h** from this set to use!

You can think of it like a game:

1. You announce your set of hash functions, **H**.
2. The adversary chooses **n** items for your hash function to hash.
3. You then randomly pick a hash function **h** from **H** to hash the **n** items.

INTUITION

Intuitively, the adversary can't foil a hash function that they don't yet know.

So, our strategy is to define a set of hash functions, and then we randomly choose a hash function **h** from this set to use!

You can think of it like a game:

1. You announce your set of hash functions, **H**.
2. The adversary chooses **n** items for your hash function to hash.
3. You then randomly pick a hash function **h** from **H** to hash the **n** items.

What would make a "good" set of hash functions H?

تابع درهم ساز خوب

معنی خوب بودن چیست؟

WHAT DOES "GOOD" MEAN?

Consider these two goals:

Which goal better represents what we want?

WHAT DOES "GOOD" MEAN?

Consider these two goals:

Design a set **H** = $\{h_1, h_2, h_{3,} \ldots, h_k\}$ where $h_i : U \rightarrow \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Which goal better represents what we want?

WHAT DOES "GOOD" MEAN?

Consider these two goals:

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

Which goal better represents what we want?

WHAT DOES "GOOD" MEAN?

Co... goals:

Design a set $\mathbf{H} = \{h_1,$ ...
where $h_i : U \to \{1$ ...
that if we chose a ...
and after an adv...
items t...

for any b...
its **expected** s...

...et $\mathbf{H} = \{h_1, h_2, h_3, \ldots, h_k\}$
... $U \to \{1, \ldots, n\}$ such
...ose a random **h** in **H**
...dversary chooses **n**
...$\mathbf{_2, \ldots, u_n}$} to hash,

...ny item $\mathbf{u_i}$,
...**cted** # of items in
bucket is **O(1)**

SUPER IMPORTANT:

The randomness is over the choice of hash function **h** from a set of hash functions **H**.

You should *not* think of it as if you've chosen a fixed hash function and are thinking about randomness over possible items the adversary could choose, or randomness over the n possible buckets in your table, or randomness over the M possible items, or anything like that.

Which goal ...hat we want?

WHAT DOES "GOOD" MEAN?

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \rightarrow \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Not what we want!

WHAT DOES "GOOD" MEAN?

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Not what we want!

Why is this goal not a good one?

Well, this *bad* set of hash functions (which always results in chains of length **n** in a single bucket) would meet this goal:

WHAT DOES "GOOD" MEAN?

Design a set **H** = {h_1, h_2, $h_{3,}$ …, h_k} where $h_i : U \rightarrow \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Not what we want!

Why is this goal not a good one?

Well, this *bad* set of hash functions (which always results in chains of length **n** in a single bucket) would meet this goal:

H = {$\mathbf{h_1}$, $\mathbf{h_2}$, …, $\mathbf{h_n}$} where
h_i maps all elements to bucket i.

WHAT DOES "GOOD" MEAN?

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Not what we want!

Why is this goal not a good one?

Well, this *bad* set of hash functions (which always results in chains of length **n** in a single bucket) would meet this goal:

$\mathbf{H = \{h_1, h_2, \ldots, h_n\}}$ where
h_i maps all elements to bucket i.

- With prob. 1/n, all **n** elements land in bucket 1
- With prob. 1/n, all **n** elements land in bucket 2
- With prob. 1/n, all **n** elements land in bucket 3

…

- With prob. 1/n, all **n** elements land in bucket n

WHAT DOES "GOOD" MEAN?

Design a set **H** = $\{h_1, h_2, h_{3,} \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Not what we want!

Why is this goal not a good one?

Well, this *bad* set of hash functions (which always results in chains of length **n** in a single bucket) would meet this goal:

$$\mathbf{H = \{h_1, h_2, \ldots, h_n\}} \text{ where}$$
$$h_i \text{ maps all elements to bucket i.}$$

- With prob. 1/n, all **n** elements land in bucket 1
- With prob. 1/n, all **n** elements land in bucket 2
- With prob. 1/n, all **n** elements land in bucket 3

…

- With prob. 1/n, all **n** elements land in bucket n

Then, **E[# items in bucket i] = 1 = O(1)** for all i…

Bucket i has **n** elements with prob. 1/n, and 0 elements with prob. (n-1)/n

WHAT DOES "GOOD" MEAN?

Consider these two goals:

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items to hash,

for any bucket,
its **expected** size is **O(1)**

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$ such that if we chose a random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1, u_2, \ldots, u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

We want the one on the right! It tries to control the expected number of collisions (which is what contributes to the linked-list traversal runtime)

WHAT DOES "GOOD" MEAN?

An analogy to explain the difference between the two:

Suppose a university offers 10 classes. 9 classes have only 1 student in them, and 1 class has 491 students.
Using the reasoning on the left, the university might say "Average class size is 50!" but in reality, it should instead report class sizes experienced by the average student (~482).

Desig
whe
that i
and a

for any bucket,
its **expected** size is **O(1)**

$_3, \ldots, h_k\}$
n} such
m **h** in **H**
hooses **n**
to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

We want the one on the right! It tries to control the expected number of collisions (which is what contributes to the linked-list traversal runtime)

WHAT WE WANT

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$, such that if we chose a random **h** in **H** and after an adversary chooses **n** items $\mathbf{\{u_1, u_2, \ldots, u_n\}}$ to hash,

for any item $\mathbf{u_i}$,
the ***expected*** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

Let's see an example of a set of hash functions H that achieves this goal!

H = EXHAUSTIVE SET OF ALL HASH FNs

WHAT WE WANT:

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \rightarrow \{1, \ldots, n\}$, such that if we chose a uniformly random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

H = EXHAUSTIVE SET OF ALL HASH FNs

WHAT WE WANT:

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$, such that if we chose a uniformly random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

H = EXHAUSTIVE SET OF ALL HASH FNs

WHAT WE WANT:

Design a set **H** = $\{h_1, h_2, h_{3,} \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$, such that if we chose a uniformly random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

Here is an example where **U** = {"a", "b", "c"} so **M = 3.** Also, we have **n = 2**.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
"a"	0	0	0	0	1	1	1	1
"b"	0	0	1	1	0	0	1	1
"c"	0	1	0	1	0	1	0	1

H = EXHAUSTIVE SET OF ALL HASH FNs

WHAT WE WANT:

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \rightarrow \{1, \ldots, n\}$, such that if we chose a uniformly random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

Here is an example where **U** = {"a", "b", "c"} so **M = 3.** Also, we have **n = 2**.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
"a"	0	0	0	0	1	1	1	1
"b"	0	0	1	1	0	0	1	1
"c"	0	1	0	1	0	1	0	1

The 0's and 1's represent the buckets i.e. h_8 will hash "b" to bucket 1.

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] =$$

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] = \sum_{j=1}^{n} P[h(u_i) = h(u_j)]$$

This probability is taken over the random choice of hash function!

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] = \sum_{j=1}^{n} P[h(u_i) = h(u_j)]$$

$$= P[h(u_i) = h(u_i)] + \sum_{j \neq i} P[h(u_i) = h(u_j)]$$

This probability is taken over the random choice of hash function!

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$\begin{aligned}
\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] &= \sum_{j=1}^{n} P[h(u_i) = h(u_j)] \\
&= P[h(u_i) = h(u_i)] + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} P[h(u_i) = h(u_j)]
\end{aligned}$$

This probability is taken over the random choice of hash function!

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$
\begin{aligned}
\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] &= \sum_{j=1}^{n} P[h(u_i) = h(u_j)] \\
&= P[h(u_i) = h(u_i)] + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} \frac{1}{n}
\end{aligned}
$$

This probability is taken over the random choice of hash function!

How do we know that **$P[h(u_i) = h(u_j)] = 1/n$** ?

You can think about it!

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$\begin{aligned}
\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] &= \sum_{j=1}^{n} P[h(u_i) = h(u_j)] \\
&= P[h(u_i) = h(u_i)] + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} \frac{1}{n} \\
&= 1 + \frac{n-1}{n}
\end{aligned}$$

This probability is taken over the random choice of hash function!

How do we know that **$P[h(u_i) = h(u_j)] = 1/n$** ?

You can think about it!

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$
\begin{aligned}
\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] &= \sum_{j=1}^{n} P[h(u_i) = h(u_j)] \\
&= P[h(u_i) = h(u_i)] + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} \frac{1}{n} \\
&= 1 + \frac{n-1}{n} \leq 2
\end{aligned}
$$

How do we know that **$P[h(u_i) = h(u_j)] = 1/n$** ?

You can think about it!

O(1)

This is what we wanted!

H = EXHAUSTIVE SET OF ALL HASH FNs

H = the exhaustive set of all hash functions that map elements in the universe U to

If the hash function we use is chosen randomly from the exhaustive set of all hash functions, then on expectation, every time we visit a bucket during an operation, there will be O(1) other things that could have also collided there!

(on avg, each student would find O(1) other students in the course!)

$$= 1 + \sum_{j \neq i} \frac{1}{n}$$

$$= 1 + \frac{n-1}{n} \leq 2$$

O(1)

This is what we wanted!

GOOD NEWS!

WHAT WE WANT:

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \rightarrow \{1, \ldots, n\}$, such that if we chose a uniformly random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

H achieves our goal! If we choose a *uniformly random hash function,* then INSERT/DELETE/SEARCH on any **n** elements will have **expected runtime of O(1)**.

سوال؟

BAD NEWS

WHAT WE WANT:

Design a set **H** = $\{h_1, h_2, h_{3,} \ldots, h_k\}$ where $h_i : U \rightarrow \{1, \ldots, n\}$, such that if we chose a uniformly random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

BAD NEWS

WHAT WE WANT:

Design a set **H** = $\{h_1, h_2, h_3, \ldots, h_k\}$ where $h_i : U \to \{1, \ldots, n\}$, such that if we chose a uniformly random **h** in **H** and after an adversary chooses **n** items $\{\mathbf{u_1, u_2, \ldots, u_n}\}$ to hash,

for any item $\mathbf{u_i}$,
the **expected** # of items in $\mathbf{u_i}$'s bucket is **O(1)**

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

How many bits does it take to store a uniformly random hash function?

A lot!

BAD NEWS

How many bits does it take to store a uniformly random hash function?

We'd use a lookup table: one entry per element of U, each storing which bucket to hash that element to.

(**M** elements) * (**log(n)** bits to write down a bucket #) = **M log n** bits

This is HUGE… (& enough to do direct addressing!)

Another way to see this:

There are $\mathbf{n^M}$ total hash functions. To uniquely identify every single hash function (each one *is* indeed unique), you'd need $\mathbf{n^M}$ different identifiers. Thus, a single identifier would take up $\mathbf{\log(n^M) = M \log n}$ bits.

BAD NEWS

How many bits does it take to store a uniformly random hash function?

We'd use a lookup table: one entry per element of U, each storing which

(**M** el **n** bits

Th g!)

How do we fix this size issue?

There are $\mathbf{n^M}$ total hash functions. To uniquely identify every single hash function (each one *is* indeed unique), you'd need $\mathbf{n^M}$ different identifiers. Thus, a single identifier would take up $\mathbf{\log(n^M) = M \log n}$ bits.

خانواده درهم سازی سراسری

مجموعه ای خوب از توابع درهم سازی که خیلی هم بزرگ نیست!

WHAT WE WANTED

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions.

$$\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] = \sum_{j=1}^{n} P[h(u_i) = h(u_j)]$$

$$= P[h(u_i) = h(u_i)] + \sum_{j \neq i} P[h(u_i) = h(u_j)]$$

$$= 1 + \sum_{j \neq i} P[h(u_i) = h(u_j)]$$

$$= 1 + \sum_{j \neq i} \frac{1}{n}$$

$$= 1 + \frac{n-1}{n} \leq 2$$

The fact that **$P[h(u_i)=h(u_j)] = 1/n$** did all the work here

O(1)

This is what we wanted!

WHAT WE WANTED

H = the exhaustive set of all hash functions that map elements in the universe U to buckets 1 to n. **H** contains a total of n^M hash functions

The exhaustive set of all hash functions achieved our goal but was way too big, so let's pick **h** from a ***smaller* hash family** where

$$\mathbf{P[h(u_i) = h(u_j)] \leq 1/n}$$

The fac

P[h(u$_i$)=h(

did all the w

$$\sum_{j \neq i} \frac{}{n}$$

$$= 1 + \frac{n-1}{n} \leq 2$$

O(1)

This is what we wanted!

UNIVERSAL HASH FAMILY

A **hash family** is a fancy name for a set of hash functions.

UNIVERSAL HASH FAMILY

A **hash family** is a fancy name for a set of hash functions.

A hash family **H** is a **universal hash family** if,
when **h** is chosen uniformly at random from **H**,

$$\text{for all } u_i, u_j \in U \text{ with } u_i \neq u_j,$$

$$P_{h \in H}\left[h(u_i) = h(u_j)\right] \leq \frac{1}{n}$$

UNIVERSAL HASH FAMILY

A **hash family** is a fancy name for a set of hash functions.

A hash family **H** is a **universal hash family** if,
when **h** is chosen uniformly at random from **H**,

$$\text{for all } u_i, u_j \in U \text{ with } u_i \neq u_j,$$

$$P_{h \in H}\left[h(u_i) = h(u_j)\right] \leq \frac{1}{n}$$

Then if we randomly choose **h** from a universal hash family **H**, we'll be guaranteed that:

E[# of items in u_i's bucket] ≤ 2 = O(1)

(FLASHBACK OF THE MATH)

A hash family **H** is a **universal hash family** if,
when **h** is chosen uniformly at random from **H**,

for all $u_i, u_j \in U$ with $u_i \neq u_j$,
$$P_{h \in H}\left[h(u_i) = h(u_j)\right] \leq \frac{1}{n}$$

$$\begin{aligned}
\mathbb{E}[\text{\# of items in } u_i \text{ 's bucket}] &= \sum_{j=1}^{n} P[h(u_i) = h(u_j)] \\
&= P[h(u_i) = h(u_i)] + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&= 1 + \sum_{j \neq i} P[h(u_i) = h(u_j)] \\
&\leq 1 + \sum_{j \neq i} \frac{1}{n} \\
&= 1 + \frac{n-1}{n} \leq 2
\end{aligned}$$

This inequality is now what a universal hash family guarantees!

O(1)

This is what we wanted!

A SMALL UNIVERSAL HASH FAMILY?

Our **H** = exhaustive set of all hash functions is a universal hash family!

It is a universal hash family, but unfortunately, as we saw earlier, this **H** is very very large. Are there smaller ones universal hash families?

A NON-EXAMPLE

$\mathbf{H = \{h_0, h_1\}}$ where

$\mathbf{h_0}$ = MOST_SIGNIFICANT_DIGIT

$\mathbf{h_1}$ = LEAST_SIGNIFICANT_DIGIT

Why is this not a universal hash family?

A NON-EXAMPLE

H = {h_0, h_1} where

h_0 = MOST_SIGNIFICANT_DIGIT

h_1 = LEAST_SIGNIFICANT_DIGIT

Why is this not a universal hash family?

$$P_{h \in H}\left[h(153) = h(173)\right] = 1 > \frac{1}{n}$$

A NON-EXAMPLE

H = {$\mathbf{h_0}$, $\mathbf{h_1}$} where

$\mathbf{h_0}$ = MOST_SIGNIFICANT_DIGIT

$\mathbf{h_1}$ = LEAST_SIGNIFICANT_DIGIT

Why is this not a universal hash family?

$$P_{h \in H}\left[h(153) = h(173)\right] = 1 > \frac{1}{n}$$

There's a ½ probability of choosing **$\mathbf{h_0}$**, and **$\mathbf{h_0}$(153) = $\mathbf{h_0}$(173) = bucket 1**

There's a ½ probability of choosing **$\mathbf{h_1}$**, and **$\mathbf{h_1}$(153) = $\mathbf{h_1}$(173) = bucket 3**

Probability that a randomly chosen **h** from **H** collides 153 & 173 is 1!

AN EXAMPLE

Here is one of the more well-studied universal hash families:

Pick a prime $\mathbf{p \geq M}$

Define $\mathbf{h_{a,b}(x) = ((ax + b) \bmod p) \bmod n}$

$$\mathbf{H = \{ h_{a,b} : a \in \{1, \ldots, p - 1\}, b \in \{0, \ldots, p - 1\} \}}$$

AN EXAMPLE

Here is one of the more well-studied universal hash families:

Pick a prime $\mathbf{p \geq M}$

Define $\mathbf{h_{a,b}(x) = ((ax + b) \bmod p) \bmod n}$

$$\mathbf{H = \{ h_{a,b} : a \in \{1, \ldots, p - 1\}, b \in \{0, \ldots, p - 1\} \}}$$

Example: Suppose n = 3, and p = 5. Here's $\mathbf{h_{2,4}}$:

$\mathbf{h_{2,4}(1) =}$ ((2*1 + 4) mod 5) mod 3 = (6 mod 5) mod 3 = 1 mod 3 = **1**

$\mathbf{h_{2,4}(4) =}$ ((2*4 + 4) mod 5) mod 3 = (12 mod 5) mod 3 = 2 mod 3 = **2**

$\mathbf{h_{2,4}(3) =}$ ((2*3 + 4) mod 5) mod 3 = (6 mod 5) mod 3 = 1 mod 3 = **1**

AN EXAMPLE

Here is one of the more well-studied universal hash families:

Pick a prime $\mathbf{p \geq M}$

Define $\mathbf{h_{a,b}(x) = ((ax + b) \bmod p) \bmod n}$

$$\mathbf{H = \{ h_{a,b} : a \in \{1, \ldots, p - 1\}, b \in \{0, \ldots, p - 1\} \}}$$

To draw a hash function **h** from **H**:

Pick a random **a** in $\{1, \ldots, p - 1\}$. & Pick a random **b** in $\{0, \ldots, p - 1\}$.

AN EXAMPLE

Here is one of the more well-studied universal hash families:

To store your $\mathbf{h_{a,b}}$, you just need to store two numbers: **a** and **b**!

Since **a** and **b** are at most p-1, we need **~2·log(p) bits**.

p is a prime that's close-ish to M, so this means the space needed =

O(log M)

This is so much better than O(M log n)!

Pick a random **a** in {1, …, p - 1}.

&

Pick a random **b** in {0, …, p - 1}.

AN EXAMPLE

Claim: This **H** is a universal hash family!

The proof is a bit complicated, and relies on number theory. See CLRS (Theorem 11.5) for details if you're curious, but **YOU ARE NOT RESPONSIBLE** for the proof in this class.

What you should know:

There exists a small universal hash family! A hash function from this universal hash family is quick to compute, lightweight to store, and relies on number theory to achieve our expected O(1) operation costs!

سوال؟

جدول درهم سازی

جمع بندی مطالب درهم سازی و استفاده عملی از آن!

THE WHOLE SCHEME

You choose your set of hash functions **H**, a universal hash family like H = mod p mod n.

H

THE WHOLE SCHEME

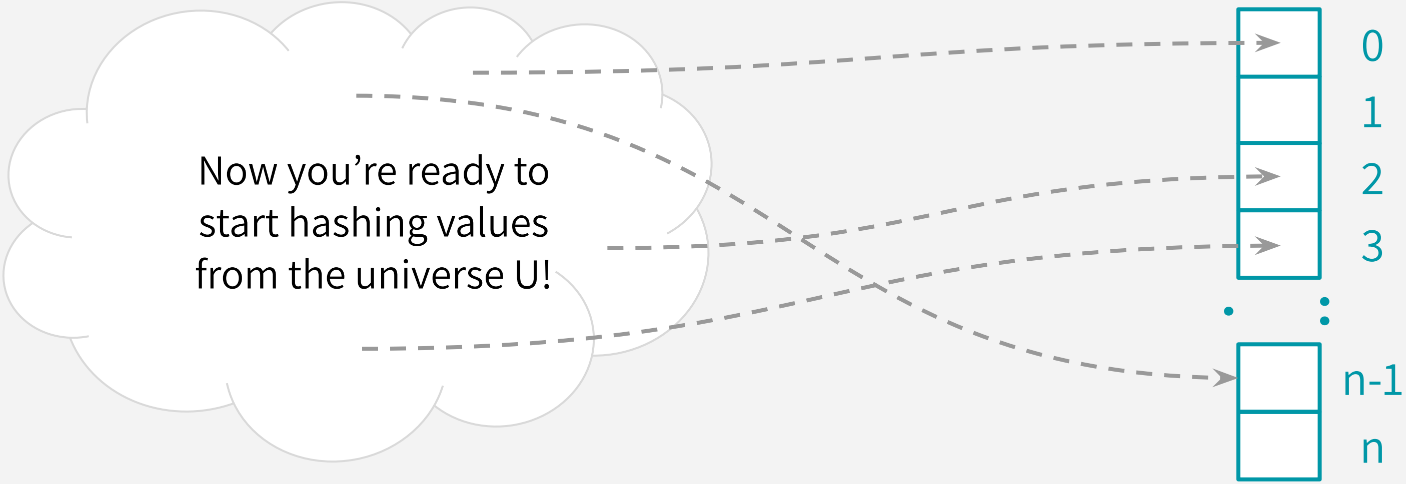
You choose your set of hash functions **H**, a universal hash family like H = mod p mod n.

When the client initializes a hash table, randomly pick a hash function **h** from **H** to use in the hash table to hash the items.

THE WHOLE SCHEME

You choose your set of hash functions **H**, a universal hash family like H = mod p mod n.

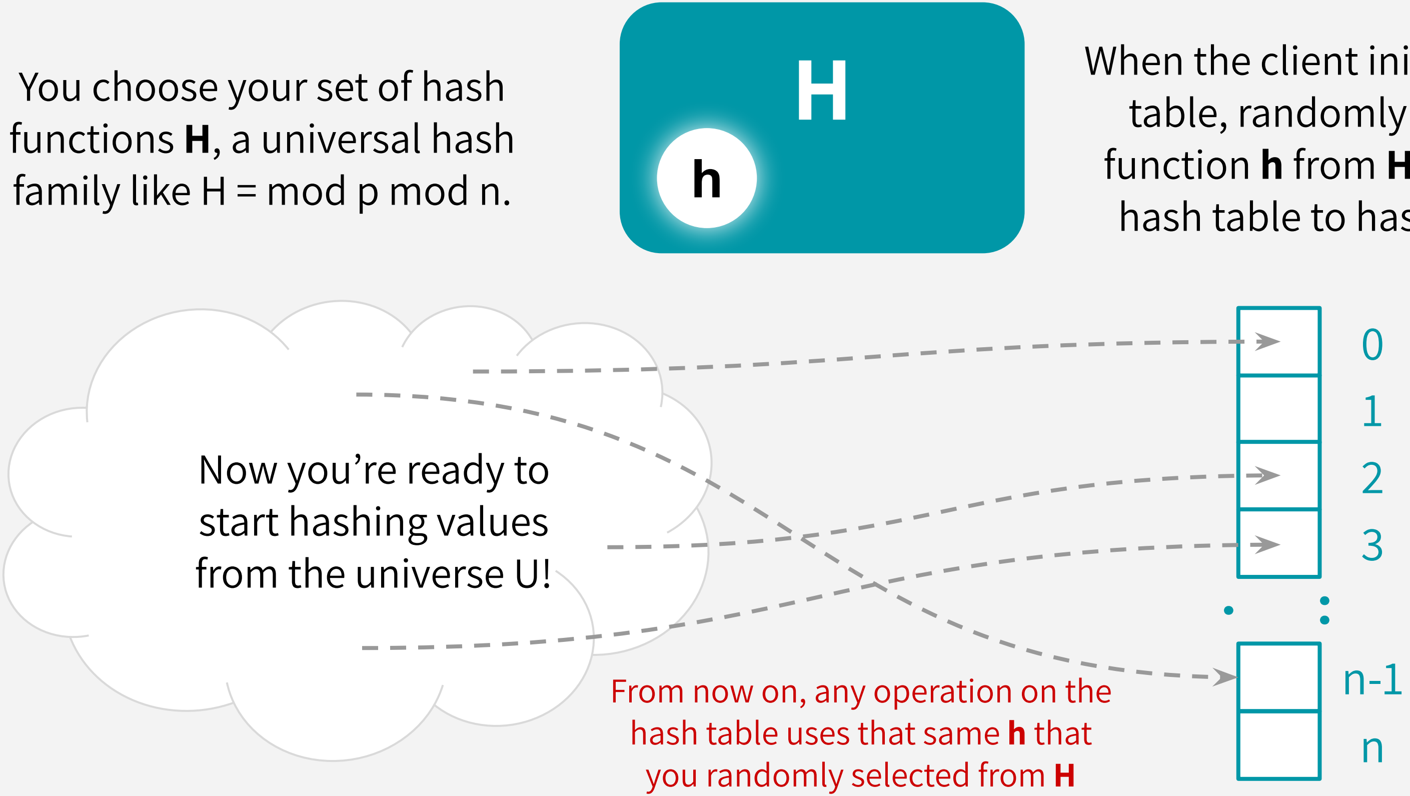
When the client initializes a hash table, randomly pick a hash function **h** from **H** to use in the hash table to hash the items.



THE WHOLE SCHEME

You choose your set of hash functions **H**, a universal hash family like H = mod p mod n.

When the client initializes a hash table, randomly pick a hash function **h** from **H** to use in the hash table to hash the items.



We can now expect that these buckets will be pretty balanced

From now on, any operation on the hash table uses that same **h** that you randomly selected from **H**

HASH TABLE MOTIVATION

OPERATION	SORTED ARRAY	UNSORTED LINKED LIST	HASH TABLES (WORST-CASE)	HASH TABLES (EXPECTED)*
SEARCH	O(log(n))	O(n)	O(n)	O(1)
DELETE	O(n)	O(n)	O(n)	O(1)
INSERT	O(n)	O(1)	O(1)	O(1)

*** Assuming we implement it cleverly with a "good" hash function**

RECAP OF HASHING

- We want a data structure that supports ***fast*** **INSERT/SEARCH/DELETE**

RECAP OF HASHING

- We want a data structure that supports ***fast*** **INSERT/SEARCH/DELETE**
- We considered this setting:
  - Come up with a set of hash functions (a hash family)
  - Bad guy chooses any n items from U & some series of operations
  - You randomly choose a hash function from your set to use

RECAP OF HASHING

- We want a data structure that supports ***fast*** **INSERT/SEARCH/DELETE**
- We considered this setting:
  - Come up with a set of hash functions (a hash family)
  - Bad guy chooses any n items from U & some series of operations
  - You randomly choose a hash function from your set to use
- **UNIVERSAL HASH FAMILIES**: a “GOOD” hash family

A hash family **H** is a **universal hash family** if, when **h** is chosen uniformly at random from **H**,

$$\text{for all } u_i, u_j \in U \text{ with } u_i \neq u_j,$$
$$P_{h\in H}\left[h(u_i) = h(u_j)\right] \leq \frac{1}{n}$$

RECAP OF HASHING

- We want a data structure that supports ***fast*** **INSERT/SEARCH/DELETE**
- We considered this setting:
  - Come up with a set of hash functions (a hash family)
  - Bad guy chooses any n items from U & some series of operations
  - You randomly choose a hash function from your set to use
- **UNIVERSAL HASH FAMILIES**: a “GOOD” hash family
  - H = exhaustive set of all hash functions
    - Good because it is a universal hash family
    - Bad because you need so much space!

A hash family **H** is a **universal hash family** if, when **h** is chosen uniformly at random from **H**,

$$\text{for all } u_i, u_j \in U \text{ with } u_i \neq u_j, \quad P_{h\in H}\left[h(u_i) = h(u_j)\right] \leq \frac{1}{n}$$

RECAP OF HASHING

- We want a data structure that supports ***fast*** **INSERT/SEARCH/DELETE**
- We considered this setting:
  - Come up with a set of hash functions (a hash family)
  - Bad guy chooses any n items from U & some series of operations
  - You randomly choose a hash function from your set to use
- **UNIVERSAL HASH FAMILIES**: a "GOOD" hash family
  - H = exhaustive set of all hash functions
    - Good because it is a universal hash family
    - Bad because you need so much space!
  - $H = \{\{ h_{a,b} : a \in \{1, \ldots, p-1\}, b \in \{0, \ldots, p-1\} \}\}$ where $h_{a,b}(x) = ((ax + b) \bmod p) \bmod n$
    - Good because it is still a universal hash family!!! (& quick to compute)
    - Good because storing an $h_{a,b}$ doesn't take up much space!!!

A hash family **H** is a **universal hash family** if, when **h** is chosen uniformly at random from **H**,

for all $u_i, u_j \in U$ with $u_i \neq u_j$,

$$P_{h \in H}\left[h(u_i) = h(u_j)\right] \leq \frac{1}{n}$$

RECAP OF HASHING

- We want a data structure that supports ***fast*** **INSERT/SEARCH/DELETE**
-
-

CONCLUSION:

We can build a hash table that supports INSERT/DELETE/SEARCH in **O(1) expected time**.

Requires **O(n log M)** bits of space:

- O(n) buckets
- O(n) items with log(M) bits per item
- O(log(M)) to store the hash function

f,

H,

- $H = \{\{ h_{a,b} : a \in \{1, \dots, p - 1\}, b \in \{0, \dots, p - 1\} \}\}$ where $h_{a,b}(x) = ((ax + b) \bmod p) \bmod n$
  - Good because it is still a universal hash family!!! (& quick to compute)
  - Good because storing an $h_{a,b}$ doesn't take up much space!!!

سوال؟