

Analysis of Idealized Open Address Hashing

$$E \left[\frac{1}{m} \sum_i X_i \right] = \frac{1}{m} \sum_i E[X_i]$$

$$\frac{n-1}{n} \cdot 1 + \frac{1}{n} \cdot 2$$

Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Let k be the $i + 1$ -st element. The expected time for a search for k is at most $\frac{1}{1-i/n} = \frac{n}{n-i}$.

$$E[X_i] \leq \frac{n}{n-i}$$

Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Let k be the $i + 1$ -st element. The expected time for a search for k is at most $\frac{1}{1-i/n} = \frac{n}{n-i}$.

$$\frac{1}{m} \sum_{i=0}^{m-1} \frac{n}{n-i}$$

Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Let k be the $i + 1$ -st element. The expected time for a search for k is at most $\frac{1}{1-i/n} = \frac{n}{n-i}$.

$$\frac{1}{m} \sum_{i=0}^{m-1} \frac{n}{n-i} = \frac{n}{m} \sum_{i=0}^{m-1} \frac{1}{n-i}$$

Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Let k be the $i + 1$ -st element. The expected time for a search for k is at most $\frac{1}{1-i/n} = \frac{n}{n-i}$.

$$\frac{1}{m} \sum_{i=0}^{m-1} \frac{n}{n-i} = \frac{n}{m} \sum_{i=0}^{m-1} \frac{1}{n-i} = \frac{1}{\alpha} \sum_{k=n-m+1}^n \frac{1}{k}$$

Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Let k be the $i + 1$ -st element. The expected time for a search for k is at most $\frac{1}{1-i/n} = \frac{n}{n-i}$.

$$\begin{aligned} \frac{1}{m} \sum_{i=0}^{m-1} \frac{n}{n-i} &= \frac{n}{m} \sum_{i=0}^{m-1} \frac{1}{n-i} = \frac{1}{\alpha} \sum_{k=n-m+1}^n \frac{1}{k} \\ &\leq \frac{1}{\alpha} \int_{n-m}^n \frac{1}{x} dx \end{aligned}$$

Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Let k be the $i + 1$ -st element. The expected time for a search for k is at most $\frac{1}{1-i/n} = \frac{n}{n-i}$.

$$\begin{aligned}\frac{1}{m} \sum_{i=0}^{m-1} \frac{n}{n-i} &= \frac{n}{m} \sum_{i=0}^{m-1} \frac{1}{n-i} = \frac{1}{\alpha} \sum_{k=n-m+1}^n \frac{1}{k} \\ &\leq \frac{1}{\alpha} \int_{n-m}^n \frac{1}{x} dx = \frac{1}{\alpha} \ln \frac{n}{n-m}\end{aligned}$$

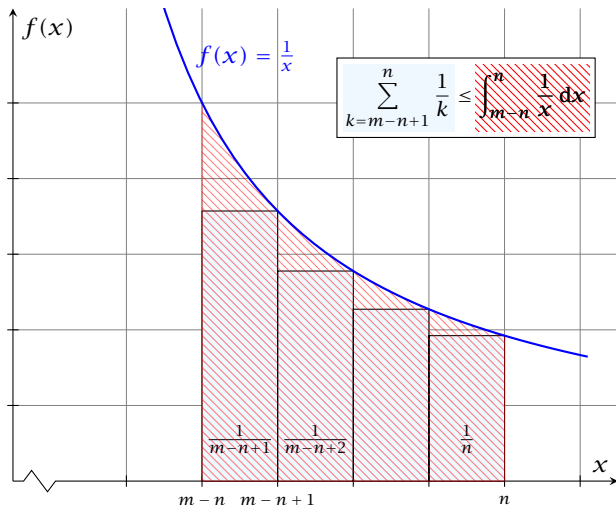
Analysis of Idealized Open Address Hashing

The number of probes in a **successful** search for k is equal to the number of probes made in an unsuccessful search for k at the time that k is inserted.

Let k be the $i + 1$ -st element. The expected time for a search for k is at most $\frac{1}{1-i/n} = \frac{n}{n-i}$.

$$\begin{aligned} \frac{1}{m} \sum_{i=0}^{m-1} \frac{n}{n-i} &= \frac{n}{m} \sum_{i=0}^{m-1} \frac{1}{n-i} = \frac{1}{\alpha} \sum_{k=n-m+1}^n \frac{1}{k} \\ &\leq \frac{1}{\alpha} \int_{n-m}^n \frac{1}{x} dx = \frac{1}{\alpha} \ln \frac{n}{n-m} = \frac{1}{\alpha} \ln \frac{1}{1-\alpha} . \end{aligned}$$

Analysis of Idealized Open Address Hashing



How do we delete in a hash-table?

- ▶ For hashing with chaining this is not a problem. Simply search for the key, and delete the item in the corresponding list.
- ▶ For open addressing this is difficult.

Deletions in Hashtables

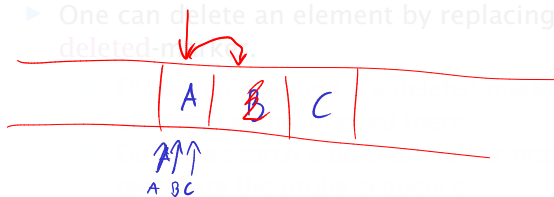
How do we delete in a hash-table?

- ▶ For hashing with chaining this is not a problem. Simply search for the key, and delete the item in the corresponding list.
- ▶ For open addressing this is difficult.

Deletions in Hashtables

- ▶ Simply removing a key might interrupt the probe sequence of other keys which then cannot be found anymore.

- ▶ One can delete an element by replacing it with a deleted-marker.



- ▶ The table could fill up with deleted-markers leading to bad performance.
- ▶ If a table contains many deleted-markers (linear fraction of the keys) one can rehash the whole table and amortize the cost for this rehash against the cost for the deletions.

Deletions in Hashtables

- ▶ Simply removing a key might interrupt the probe sequence of other keys which then cannot be found anymore.
- ▶ One can delete an element by replacing it with a **deleted**-marker.
 - ▶ During an insertion if a deleted-marker is encountered an element can be inserted there.
 - ▶ During a search a deleted-marker must not be used to terminate the probe sequence.
- ▶ The table could fill up with deleted-markers leading to bad performance.
- ▶ If a table contains many deleted-markers (linear fraction of the keys) one can rehash the whole table and amortize the cost for this rehash against the cost for the deletions.

Deletions in Hashtables

- ▶ Simply removing a key might interrupt the probe sequence of other keys which then cannot be found anymore.
- ▶ One can delete an element by replacing it with a **deleted**-marker.
 - ▶ During an insertion if a **deleted**-marker is encountered an element can be inserted there.
 - ▶ During a search a **deleted**-marker must not be used to terminate the probe sequence.
- ▶ The table could fill up with **deleted**-markers leading to bad performance.
- ▶ If a table contains many **deleted**-markers (linear fraction of the keys) one can rehash the whole table and amortize the cost for this rehash against the cost for the deletions.

Deletions in Hashtables

- ▶ Simply removing a key might interrupt the probe sequence of other keys which then cannot be found anymore.
- ▶ One can delete an element by replacing it with a **deleted**-marker.
 - ▶ During an insertion if a **deleted**-marker is encountered an element can be inserted there.
 - ▶ During a search a **deleted**-marker must not be used to terminate the probe sequence.
- ▶ The table could fill up with **deleted**-markers leading to bad performance.
- ▶ If a table contains many **deleted**-markers (linear fraction of the keys) one can rehash the whole table and amortize the cost for this rehash against the cost for the deletions.

Deletions in Hashtables

- ▶ Simply removing a key might interrupt the probe sequence of other keys which then cannot be found anymore.
- ▶ One can delete an element by replacing it with a **deleted**-marker.
 - ▶ During an insertion if a **deleted**-marker is encountered an element can be inserted there.
 - ▶ During a search a **deleted**-marker must not be used to terminate the probe sequence.
- ▶ The table could fill up with **deleted**-markers leading to bad performance.
- ▶ If a table contains many deleted-markers (linear fraction of the keys) one can rehash the whole table and amortize the cost for this rehash against the cost for the deletions.

Deletions in Hashtables

- ▶ Simply removing a key might interrupt the probe sequence of other keys which then cannot be found anymore.
- ▶ One can delete an element by replacing it with a **deleted**-marker.
 - ▶ During an insertion if a **deleted**-marker is encountered an element can be inserted there.
 - ▶ During a search a **deleted**-marker must not be used to terminate the probe sequence.
- ▶ The table could fill up with **deleted**-markers leading to bad performance.
- ▶ If a table contains many deleted-markers (linear fraction of the keys) one can rehash the whole table and amortize the cost for this rehash against the cost for the deletions.

Deletions for Linear Probing

- ▶ For Linear Probing one can delete elements without using **deletion**-markers.
- ▶ Upon a deletion elements that are further down in the probe-sequence may be moved to guarantee that they are still found during a search.

Deletions for Linear Probing

- ▶ For Linear Probing one can delete elements without using **deletion**-markers.
- ▶ Upon a deletion elements that are further down in the probe-sequence may be moved to guarantee that they are still found during a search.

Deletions for Linear Probing

Algorithm 12 delete(p)

```
1:  $T[p] \leftarrow \text{null}$ 
2:  $p \leftarrow \text{succ}(p)$ 
3: while  $T[p] \neq \text{null}$  do
4:    $y \leftarrow T[p]$ 
5:    $T[p] \leftarrow \text{null}$ 
6:    $p \leftarrow \text{succ}(p)$ 
7:    $\text{insert}(y)$ 
```

p

p is the index into the table-cell that contains the object to be deleted.

Pointers into the hash-table become invalid.

Deletions for Linear Probing

Algorithm 12 delete(p)

```
1:  $T[p] \leftarrow \text{null}$ 
2:  $p \leftarrow \text{succ}(p)$ 
3: while  $T[p] \neq \text{null}$  do
4:    $y \leftarrow T[p]$ 
5:    $T[p] \leftarrow \text{null}$ 
6:    $p \leftarrow \text{succ}(p)$ 
7:    $\text{insert}(y)$ 
```

p is the index into the table-cell that contains the object to be deleted.

Pointers into the hash-table become invalid.

Universal Hashing

Regardless, of the choice of hash-function there is always an input (a set of keys) that has a very poor worst-case behaviour.

Therefore, so far we assumed that the hash-function is random so that regardless of the input the average case behaviour is good.

However, the assumption of uniform hashing that h is chosen randomly from all functions $f: U \rightarrow [0, \dots, n-1]$ is clearly unrealistic as there are $n^{|U|}$ such functions. Even writing down such a function would take $|U| \log n$ bits.

Universal hashing tries to define a set \mathcal{H} of functions that is much smaller but still leads to good average case behaviour when selecting a hash-function uniformly at random from \mathcal{H} .

Universal Hashing

Regardless, of the choice of hash-function there is always an input (a set of keys) that has a very poor worst-case behaviour.

Therefore, so far we assumed that the hash-function is random so that regardless of the input the average case behaviour is good.

However, the assumption of uniform hashing that h is chosen randomly from all functions $f : U \rightarrow [0, \dots, n - 1]$ is clearly unrealistic as there are $n^{|U|}$ such functions. Even writing down such a function would take $|U| \log n$ bits.

Universal hashing tries to define a set \mathcal{H} of functions that is much smaller but still leads to good average case behaviour when selecting a hash-function uniformly at random from \mathcal{H} .

Universal Hashing

Regardless, of the choice of hash-function there is always an input (a set of keys) that has a very poor worst-case behaviour.

Therefore, so far we assumed that the hash-function is random so that regardless of the input the average case behaviour is good.

However, the assumption of uniform hashing that h is chosen randomly from all functions $f : U \rightarrow [0, \dots, n - 1]$ is clearly unrealistic as there are $n^{|U|}$ such functions. Even writing down such a function would take $|U| \log n$ bits.

Universal hashing tries to define a set \mathcal{H} of functions that is much smaller but still leads to good average case behaviour when selecting a hash-function uniformly at random from \mathcal{H} .

Universal Hashing

Regardless, of the choice of hash-function there is always an input (a set of keys) that has a very poor worst-case behaviour.

Therefore, so far we assumed that the hash-function is random so that regardless of the input the average case behaviour is good.

However, the assumption of uniform hashing that h is chosen randomly from all functions $f : U \rightarrow [0, \dots, n - 1]$ is clearly unrealistic as there are $n^{|U|}$ such functions. Even writing down such a function would take $|U| \log n$ bits.

Universal hashing tries to define a set \mathcal{H} of functions that is much smaller but still leads to good average case behaviour when selecting a hash-function uniformly at random from \mathcal{H} .

Universal Hashing

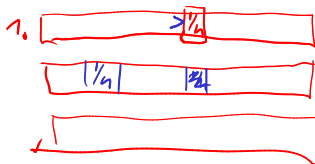
Regardless, of the choice of hash-function there is always an input (a set of keys) that has a very poor worst-case behaviour.

Therefore, so far we assumed that the hash-function is random so that regardless of the input the average case behaviour is good.

However, the assumption of uniform hashing that h is chosen randomly from all functions $f : U \rightarrow [0, \dots, n - 1]$ is clearly unrealistic as there are $n^{|U|}$ such functions. Even writing down such a function would take $|U| \log n$ bits.

Universal hashing tries to define a set \mathcal{H} of functions that is much smaller but still leads to good average case behaviour when selecting a hash-function uniformly at random from \mathcal{H} .

Universal Hashing



Definition 24

A class \mathcal{H} of hash-functions from the universe U into the set $\{0, \dots, n-1\}$ is called **universal** if for all $u_1, u_2 \in U$ with $u_1 \neq u_2$

$$\Pr[h(u_1) = h(u_2)] \leq \frac{1}{n} ,$$

where the probability is w. r. t. the choice of a random hash-function from set \mathcal{H} .

Note that this means that the probability of a collision between two arbitrary elements is at most $\frac{1}{n}$.

Universal Hashing

Definition 24

A class \mathcal{H} of hash-functions from the universe U into the set $\{0, \dots, n-1\}$ is called **universal** if for all $u_1, u_2 \in U$ with $u_1 \neq u_2$

$$\Pr[h(u_1) = h(u_2)] \leq \frac{1}{n} ,$$

where the probability is w. r. t. the choice of a random hash-function from set \mathcal{H} .

Note that this means that the probability of a collision between two arbitrary elements is at most $\frac{1}{n}$.

Universal Hashing

Definition 25

A class \mathcal{H} of hash-functions from the universe U into the set $\{0, \dots, n-1\}$ is called **2-independent** (pairwise independent) if the following two conditions hold

- ▶ For any key $u \in U$, and $t \in \{0, \dots, n-1\}$ $\Pr[h(u) = t] = \frac{1}{n}$, i.e., a key is distributed uniformly within the hash-table.
- ▶ For all $u_1, u_2 \in U$ with $u_1 \neq u_2$, and for any two hash-positions t_1, t_2 :

$$\Pr[h(u_1) = t_1 \wedge h(u_2) = t_2] \leq \frac{1}{n^2} .$$

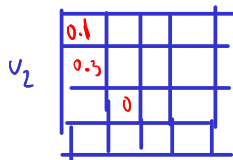
This requirement clearly implies a universal hash-function.

Universal Hashing

Definition 25

A class \mathcal{H} of hash-functions from the universe U into the set $\{0, \dots, n-1\}$ is called **2-independent** (pairwise independent) if the following two conditions hold

- ▶ For any key $u \in U$, and $t \in \{0, \dots, n-1\}$ $\Pr[h(u) = t] = \frac{1}{n}$,
i.e., a key is distributed uniformly within the hash-table.
- ▶ For all $u_1, u_2 \in U$ with $u_1 \neq u_2$, and for any two
 \forall_n hash-positions t_1, t_2 :



$$\Pr[h(u_1) = t_1 \wedge h(u_2) = t_2] \stackrel{!}{=} \frac{1}{n^2} .$$

This requirement clearly implies a universal hash-function.

Universal Hashing

Definition 26

A class \mathcal{H} of hash-functions from the universe U into the set $\{0, \dots, n-1\}$ is called **k -independent** if for any choice of $\ell \leq k$ distinct keys $u_1, \dots, u_\ell \in U$, and for any set of ℓ not necessarily distinct hash-positions t_1, \dots, t_ℓ :

$$\Pr[h(u_1) = t_1 \wedge \dots \wedge h(u_\ell) = t_\ell] \leq \frac{1}{n^\ell} ,$$

where the probability is w. r. t. the choice of a random hash-function from set \mathcal{H} .

Universal Hashing

Definition 27

A class \mathcal{H} of hash-functions from the universe U into the set $\{0, \dots, n-1\}$ is called (μ, k) -independent if for any choice of $\ell \leq k$ distinct keys $u_1, \dots, u_\ell \in U$, and for any set of ℓ not necessarily distinct hash-positions t_1, \dots, t_ℓ :

$$\Pr[h(u_1) = t_1 \wedge \dots \wedge h(u_\ell) = t_\ell] \leq \frac{\mu}{n^\ell},$$

where the probability is w. r. t. the choice of a random hash-function from set \mathcal{H} .

Universal Hashing

Let $U := \{0, \dots, p-1\}$ for a prime p . Let $\mathbb{Z}_p := \{0, \dots, p-1\}$, and let $\mathbb{Z}_p^* := \{1, \dots, p-1\}$ denote the set of invertible elements in \mathbb{Z}_p .

Define

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

Lemma 28

The class

$$\mathcal{H} = \{h_{a,b} \mid a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p\}$$

is a universal class of hash-functions from U to $\{0, \dots, n-1\}$.

Universal Hashing

Let $U := \{0, \dots, p-1\}$ for a prime p . Let $\mathbb{Z}_p := \{0, \dots, p-1\}$, and let $\mathbb{Z}_p^* := \{1, \dots, p-1\}$ denote the set of invertible elements in \mathbb{Z}_p .

Define

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

Lemma 28

The class

$$\mathcal{H} = \{h_{a,b} \mid a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p\}$$

is a universal class of hash-functions from U to $\{0, \dots, n-1\}$.

Universal Hashing

Let $U := \{0, \dots, p-1\}$ for a prime p . Let $\mathbb{Z}_p := \{0, \dots, p-1\}$, and let $\mathbb{Z}_p^* := \{1, \dots, p-1\}$ denote the set of invertible elements in \mathbb{Z}_p .

Define

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

Lemma 28

The class

$$\mathcal{H} = \{h_{a,b} \mid a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p\}$$

is a universal class of hash-functions from U to $\{0, \dots, n-1\}$.

Universal Hashing

Let $U := \{0, \dots, p-1\}$ for a prime p . Let $\mathbb{Z}_p := \{0, \dots, p-1\}$, and let $\mathbb{Z}_p^* := \{1, \dots, p-1\}$ denote the set of invertible elements in \mathbb{Z}_p .

Define

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

Lemma 28

The class

$$\mathcal{H} = \{h_{a,b} \mid a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p\}$$

is a universal class of hash-functions from U to $\{0, \dots, n-1\}$.

Universal Hashing

Proof.

Let $x, y \in U$ be two distinct keys. We have to show that the probability of a collision is only $1/n$.

Universal Hashing

Proof.

Let $x, y \in U$ be two distinct keys. We have to show that the probability of a collision is only $1/n$.

$$\blacktriangleright ax + b \neq ay + b \pmod{p}$$

Universal Hashing

Proof.

Let $x, y \in U$ be two distinct keys. We have to show that the probability of a collision is only $1/n$.

► $ax + b \not\equiv ay + b \pmod{p}$

If $x \neq y$ then $(x - y) \not\equiv 0 \pmod{p}$.

Multiplying with $a \not\equiv 0 \pmod{p}$ gives

$$a(x - y) \not\equiv 0 \pmod{p}$$

where we use that \mathbb{Z}_p is a field (Körper) and, hence, has no zero divisors (nullteilerfrei).

Universal Hashing

Proof.

Let $x, y \in U$ be two distinct keys. We have to show that the probability of a collision is only $1/n$.

► $ax + b \not\equiv ay + b \pmod{p}$

If $x \neq y$ then $(x - y) \not\equiv 0 \pmod{p}$.

Multiplying with $a \not\equiv 0 \pmod{p}$ gives

$$a(x - y) \not\equiv 0 \pmod{p}$$

where we use that \mathbb{Z}_p is a field (Körper) and, hence, has no zero divisors (nullteilerfrei).

Universal Hashing

Proof.

Let $x, y \in U$ be two distinct keys. We have to show that the probability of a collision is only $1/n$.

► $ax + b \not\equiv ay + b \pmod{p}$

If $x \neq y$ then $(x - y) \not\equiv 0 \pmod{p}$.

Multiplying with $a \not\equiv 0 \pmod{p}$ gives

$$a(x - y) \not\equiv 0 \pmod{p}$$

where we use that \mathbb{Z}_p is a field (Körper) and, hence, has no zero divisors (nullteilerfrei).

Universal Hashing

Proof.

Let $x, y \in U$ be two distinct keys. We have to show that the probability of a collision is only $1/n$.

► $ax + b \not\equiv ay + b \pmod{p}$

If $x \neq y$ then $(x - y) \not\equiv 0 \pmod{p}$.

Multiplying with $a \not\equiv 0 \pmod{p}$ gives

$$a(x - y) \not\equiv 0 \pmod{p}$$

where we use that \mathbb{Z}_p is a field (**Körper**) and, hence, has no zero divisors (**nullteilerfrei**).

Universal Hashing

- ▶ The hash-function does not generate collisions before the $(\text{mod } n)$ -operation. Furthermore, every choice (a, b) is mapped to a different pair (t_x, t_y) with $t_x := ax + b$ and $t_y := ay + b$.

Universal Hashing

- ▶ The hash-function does not generate collisions before the $(\text{mod } n)$ -operation. Furthermore, every choice (a, b) is mapped to a different pair (t_x, t_y) with $t_x := ax + b$ and $t_y := ay + b$.

This holds because we can compute a and b when given t_x and t_y :

Universal Hashing

- ▶ The hash-function does not generate collisions before the $(\text{mod } n)$ -operation. Furthermore, every choice (a, b) is mapped to a different pair (t_x, t_y) with $t_x := ax + b$ and $t_y := ay + b$.

This holds because we can compute a and b when given t_x and t_y :

$$t_x \equiv ax + b \pmod{p}$$

$$t_y \equiv ay + b \pmod{p}$$

Universal Hashing

- ▶ The hash-function does not generate collisions before the $(\text{mod } n)$ -operation. Furthermore, every choice (a, b) is mapped to a different pair (t_x, t_y) with $t_x := ax + b$ and $t_y := ay + b$.

This holds because we can compute a and b when given t_x and t_y :

$$t_x \equiv ax + b \pmod{p}$$

$$t_y \equiv ay + b \pmod{p}$$

$$t_x - t_y \equiv a(x - y) \pmod{p}$$

$$t_y \equiv ay + b \pmod{p}$$

Universal Hashing

- ▶ The hash-function does not generate collisions before the $(\text{mod } n)$ -operation. Furthermore, every choice (a, b) is mapped to a different pair (t_x, t_y) with $t_x := ax + b$ and $t_y := ay + b$.

This holds because we can compute a and b when given t_x and t_y :

$$t_x \equiv ax + b \pmod{p}$$

$$t_y \equiv ay + b \pmod{p}$$

$$t_x - t_y \equiv a(x - y) \pmod{p}$$

$$t_y \equiv ay + b \pmod{p}$$

$$a \equiv (t_x - t_y)(x - y)^{-1} \pmod{p}$$

$$b \equiv t_y - ay \pmod{p}$$

Universal Hashing

There is a one-to-one correspondence between hash-functions (pairs (a, b) , $a \neq 0$) and pairs (t_x, t_y) , $t_x \neq t_y$.

Therefore, we can view the first step (before the mod n -operation) as choosing a pair (t_x, t_y) , $t_x \neq t_y$ uniformly at random.

What happens when we do the mod n operation?

Fix a value t_x . There are $p - 1$ possible values for choosing t_y .

From the range $0, \dots, p - 1$ the values $t_x, t_x + n, t_x + 2n, \dots$ map to t_x after the modulo-operation. These are at most $\lceil p/n \rceil$ values.

Universal Hashing

There is a one-to-one correspondence between hash-functions (pairs (a, b) , $a \neq 0$) and pairs (t_x, t_y) , $t_x \neq t_y$.

Therefore, we can view the first step (before the mod n -operation) as choosing a pair (t_x, t_y) , $t_x \neq t_y$ uniformly at random.

What happens when we do the mod n operation?

Fix a value t_x . There are $p - 1$ possible values for choosing t_y .

From the range $0, \dots, p - 1$ the values $t_x, t_x + n, t_x + 2n, \dots$ map to t_x after the modulo-operation. These are at most $\lceil p/n \rceil$ values.

Universal Hashing

There is a one-to-one correspondence between hash-functions (pairs (a, b) , $a \neq 0$) and pairs (t_x, t_y) , $t_x \neq t_y$.

Therefore, we can view the first step (before the $\text{mod } n$ -operation) as choosing a pair (t_x, t_y) , $t_x \neq t_y$ uniformly at random.

What happens when we do the $\text{mod } n$ operation?

Fix a value t_x . There are $p - 1$ possible values for choosing t_y .

From the range $0, \dots, p - 1$ the values $t_x, t_x + n, t_x + 2n, \dots$ map to t_x after the modulo-operation. These are at most $\lfloor p/n \rfloor$ values.

Universal Hashing

There is a one-to-one correspondence between hash-functions (pairs (a, b) , $a \neq 0$) and pairs (t_x, t_y) , $t_x \neq t_y$.

Therefore, we can view the first step (before the $\text{mod } n$ -operation) as choosing a pair (t_x, t_y) , $t_x \neq t_y$ uniformly at random.

What happens when we do the $\text{mod } n$ operation?

Fix a value t_x . There are $p - 1$ possible values for choosing t_y .

From the range $0, \dots, p - 1$ the values $t_x, t_x + n, t_x + 2n, \dots$ map to t_x after the modulo-operation. These are at most $\lceil p/n \rceil$ values.

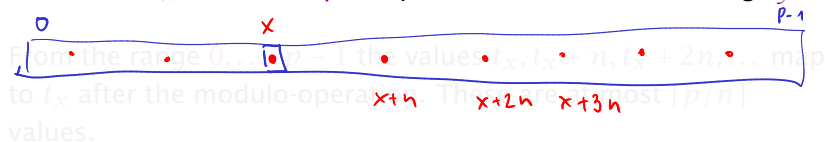
Universal Hashing

There is a one-to-one correspondence between hash-functions (pairs (a, b) , $a \neq 0$) and pairs (t_x, t_y) , $t_x \neq t_y$.

Therefore, we can view the first step (before the $\text{mod } n$ -operation) as choosing a pair (t_x, t_y) , $t_x \neq t_y$ uniformly at random.

What happens when we do the $\text{mod } n$ operation?

Fix a value t_x . There are $p - 1$ possible values for choosing t_y .



Universal Hashing

There is a one-to-one correspondence between hash-functions (pairs (a, b) , $a \neq 0$) and pairs (t_x, t_y) , $t_x \neq t_y$.

Therefore, we can view the first step (before the $\text{mod } n$ -operation) as choosing a pair (t_x, t_y) , $t_x \neq t_y$ uniformly at random.

What happens when we do the $\text{mod } n$ operation?

Fix a value t_x . There are $p - 1$ possible values for choosing t_y .

From the range $0, \dots, p - 1$ the values $t_x, t_x + n, t_x + 2n, \dots$ map to t_x after the modulo-operation. These are at most $\lceil p/n \rceil$ values.

Universal Hashing

As $t_y \neq t_x$ there are

possibilities for choosing t_y such that the final hash-value creates a collision.

This happens with probability at most $\frac{1}{n}$.

Universal Hashing

As $t_y \neq t_x$ there are

$$\left\lceil \frac{p}{n} \right\rceil - 1 \leq \frac{p}{n} + \frac{n-1}{n} - 1 \leq \frac{p-1}{n}$$

possibilities for choosing t_y such that the final hash-value creates a collision.

This happens with probability at most $\frac{1}{n}$.

Universal Hashing

As $t_y \neq t_x$ there are

$$\left\lceil \frac{p}{n} \right\rceil - 1 \leq \frac{p}{n} + \frac{n-1}{n} - 1 \leq \frac{p-1}{n}$$

possibilities for choosing t_y such that the final hash-value creates a collision.

This happens with probability at most $\frac{1}{n}$.

Universal Hashing

As $t_y \neq t_x$ there are

$$\left\lceil \frac{p}{n} \right\rceil - 1 \leq \frac{p}{n} + \frac{n-1}{n} - 1 \leq \frac{p-1}{n}$$

possibilities for choosing t_y such that the final hash-value creates a collision.

This happens with probability at most $\frac{1}{n}$.

Universal Hashing

As $t_y \neq t_x$ there are

$$\left\lceil \frac{p}{n} \right\rceil - 1 \leq \frac{p}{n} + \frac{n-1}{n} - 1 \leq \frac{p-1}{n}$$

possibilities for choosing t_y such that the final hash-value creates a collision.

This happens with probability at most $\frac{1}{n}$.

Universal Hashing

As $t_y \neq t_x$ there are

$$\left\lceil \frac{p}{n} \right\rceil - 1 \leq \frac{p}{n} + \frac{n-1}{n} - 1 \leq \frac{p-1}{n}$$

possibilities for choosing t_y such that the final hash-value creates a collision.

This happens with probability at most $\frac{1}{n}$.

Universal Hashing



Universal Hashing

It is also possible to show that \mathcal{H} is an (almost) pairwise independent class of hash-functions.

$$\Pr_{t_x \neq t_y \in \mathbb{Z}_p^2} \left[\begin{array}{l} t_x \bmod n = h_1 \\ t_y \bmod n = h_2 \end{array} \right]$$

Universal Hashing

It is also possible to show that \mathcal{H} is an (almost) pairwise independent class of hash-functions.

$$\frac{\left\lfloor \frac{p}{n} \right\rfloor^2}{p(p-1)} \leq \Pr_{t_x \neq t_y \in \mathbb{Z}_p^2} \left[\begin{array}{l} t_x \bmod n = h_1 \\ t_y \bmod n = h_2 \end{array} \right] \leq \frac{\left\lfloor \frac{p}{n} \right\rfloor^2}{p(p-1)}$$

Universal Hashing

It is also possible to show that \mathcal{H} is an (almost) pairwise independent class of hash-functions.

$$\frac{\lfloor \frac{p}{n} \rfloor^2}{p(p-1)} \leq \Pr_{t_x \neq t_y \in \mathbb{Z}_p^2} \left[\begin{array}{l} t_x \bmod n = h_1 \\ t_y \bmod n = h_2 \end{array} \right] \leq \frac{\lceil \frac{p}{n} \rceil^2}{p(p-1)}$$

Note that the middle is the probability that $h(x) = h_1$ and $h(y) = h_2$. The total number of choices for (t_x, t_y) is $p(p-1)$. The number of choices for t_x (t_y) such that $t_x \bmod n = h_1$ ($t_y \bmod n = h_2$) lies between $\lfloor \frac{p}{n} \rfloor$ and $\lceil \frac{p}{n} \rceil$.