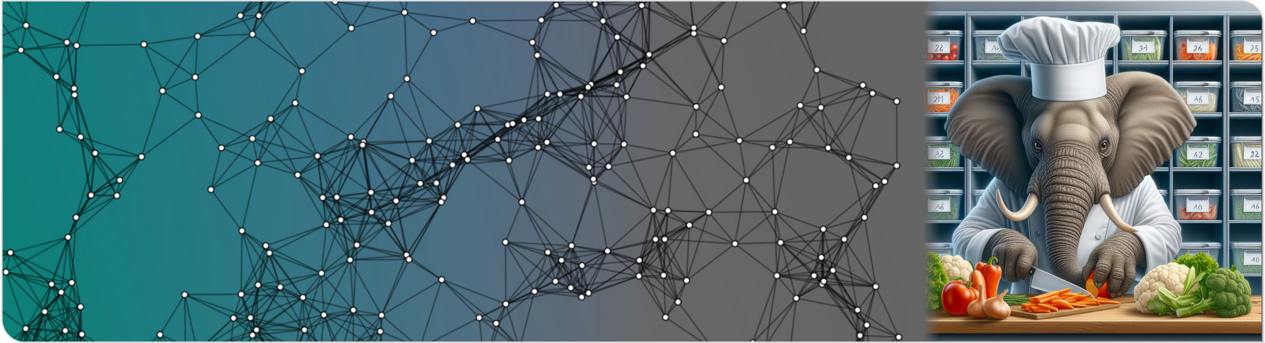
Karlsruhe Institute of Technology

Probability and Computing – Classic Hash Tables

Stefan Walzer, Maximilian Katzmann | WS 2023/2024



Prüfungsanmeldung

- Am einfachsten: Hier angeben, wann ihr Zeit habt:
  `https://www.terminplaner.dfn.de/W4m8QyA9vvp1K19m`
- Alternativ: Email an Stefan und Max.
- Wir bieten euch dann einen Termin per Email an.

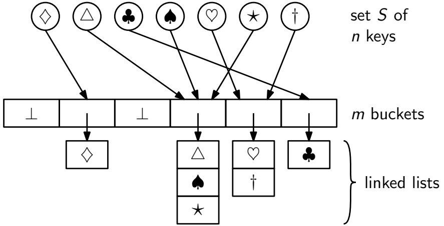
Content

Hash Table with Chaining

e.g. `std::unordered_set`, `java.util.HashMap`

Terminology

D:	*Universe* (or domain) of *keys* (strings, integers, game states in chess)
$S \subseteq D$:	set of n keys (possibly with associated data)
$h : D \to R$:	hash function, range usually $R = [m]$
$\alpha = \frac{n}{m}$:	*load factor*, $\alpha \leq \alpha_{\text{max}} = \mathcal{O}(1)$



Hash Table with Chaining

e.g. `std::unordered_set`, `java.util.HashMap`

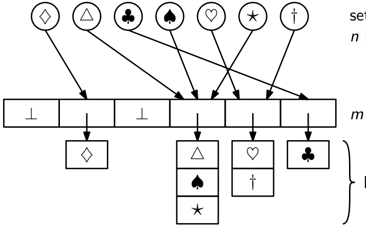
Terminology

D: *Universe* (or domain) of *keys* (strings, integers, game states in chess)

$S \subseteq D$: set of n keys (possibly with associated data)

$h : D \to R$: hash function, range usually $R = [m]$

$\alpha = \frac{n}{m}$: *load factor*, $\alpha \leq \alpha_{\max} = \mathcal{O}(1)$



Goal

Operations in time t with $\mathbb{E}[t] = \mathcal{O}(1)$.
Randomness comes from the hash function.

Hash Table with Chaining

e.g. `std::unordered_set`, `java.util.HashMap`

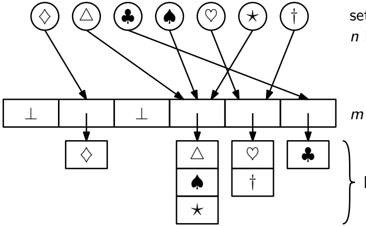
Terminology

D: *Universe* (or domain) of *keys* (strings, integers, game states in chess)

$S \subseteq D$: set of n keys (possibly with associated data)

$h : D \to R$: hash function, range usually $R = [m]$

$\alpha = \frac{n}{m}$: *load factor*, $\alpha \leq \alpha_{\max} = \mathcal{O}(1)$



Goal

Operations in time t with $\mathbb{E}[t] = \mathcal{O}(1)$.
Randomness comes from the hash function.

Ideal Hash Functions

Every function from D to R is equally likely to be h.

Ideal Hash Functions are Impractical

Karlsruhe Institute of Technology

Naive Idea

- Let R^D denote all functions from D to R. We pick $h \sim \mathcal{U}(R^D)$.
- There are $|R|$ options for the hash of each $x \in D$
- Hence: $|R^D| = |R|^{|D|}$

$x \in D$	x_1	x_2	x_3	$\dots$	$x_{\lvert D \rvert}$
$h(x) \in R$	?	?	?	$\dots$	?

Why $h \sim \mathcal{U}(R^D)$ is desirable

- $h \sim \mathcal{U}(R^D) \Leftrightarrow \forall x_1, \dots, x_n \in D : h(x_1), h(x_2), \dots, h(x_n)$ are *independent* and uniformly random in R.
  $\hookrightarrow$ independence is very useful in an analysis
- In particular: $\forall x_1, \dots, x_n \in D, \forall i_1, \dots, i_n : \Pr_{h \sim \mathcal{U}(R^D)}[h(x_1) = i_1 \wedge \dots \wedge h(x_n) = i_n] = |R|^{-n}$.

Why $h \sim \mathcal{U}(R^D)$ is unwieldy

$\log_2(|R|^{|D|}) = |D| \cdot \log_2(|R|)$ bits to store $h \sim \mathcal{U}(R^D)$ $\rightsquigarrow$ for $D = \{0,1\}^{64}$: more than 2^{64} bits.

Content

What is a Hash Function?
(it depends on who you ask)

What is a Hash Function?
(it depends on who you ask)

Cryptographic Hash Function

A **collision resistant** function such as $h =$ `sha256sum`

```
$ sha256sum myfile.txt
018a7eaee8a...3e79043e21ab4  myfile.txt
```

Range $R = \{0, 1\}^{256}$. It is hard to find x, y with $h(x) = h(y)$.
$\hookrightarrow$ Files with equal hashes are likely the same.

What is a Hash Function?

(it depends on who you ask)

Karlsruhe Institute of Technology

Cryptographic Hash Function

A **collision resistant** function such as $h =$ `sha256sum`

```
$ sha256sum myfile.txt
018a7eaee8a...3e79043e21ab4  myfile.txt
```

Range $R = \{0,1\}^{256}$. It is hard to find x, y with $h(x) = h(y)$.

$\hookrightarrow$ Files with equal hashes are likely the same.

Cryptographic Pseudorandom Function

A function $f : \text{Seeds} \times D \to R$ where $\log_2 |\text{Seeds}|$ is small and no efficient algorithm can distinguish

- $f(s, \cdot)$ for $s \sim \mathcal{U}(\text{Seeds})$ and
- $h(\cdot)$ for $h \sim \mathcal{U}(R^D)$,

except with negligible probability.

What is a Hash Function?

(it depends on who you ask)

Cryptographic Hash Function

A **collision resistant** function such as $h =$ `sha256sum`

```
$ sha256sum myfile.txt
018a7eaee8a...3e79043e21ab4  myfile.txt
```

Range $R = \{0,1\}^{256}$. It is hard to find x, y with $h(x) = h(y)$.
$\hookrightarrow$ Files with equal hashes are likely the same.

Cryptographic Pseudorandom Function

A function f : Seeds $\times D \to R$ where $\log_2 |\text{Seeds}|$ is small and no efficient algorithm can distinguish

- $f(s,\cdot)$ for $s \sim \mathcal{U}(\text{Seeds})$ and
- $h(\cdot)$ for $h \sim \mathcal{U}(R^D)$,

except with negligible probability.

Hash Function in Algorithm Engineering

- typically small range $|R| = \mathcal{O}(n)$
  $\hookrightarrow$ cannot be collision resistant
- should **behave like** $h \sim \mathcal{U}(R^D)$ **in my application**
- should be **fast** to evaluate

What is a Hash Function?

(it depends on who you ask)

Cryptographic Hash Function

A **collision resistant** function such as $h =$ `sha256sum`

```
$ sha256sum myfile.txt
018a7eaee8a...3e79043e21ab4  myfile.txt
```

Range $R = \{0,1\}^{256}$. It is hard to find x, y with $h(x) = h(y)$.

$\hookrightarrow$ Files with equal hashes are likely the same.

Cryptographic Pseudorandom Function

A function $f : \text{Seeds} \times D \to R$ where $\log_2 |\text{Seeds}|$ is small and no efficient algorithm can distinguish

- $f(s, \cdot)$ for $s \sim \mathcal{U}(\text{Seeds})$ and
- $h(\cdot)$ for $h \sim \mathcal{U}(R^D)$,

except with negligible probability.

Hash Function in Algorithm Engineering

- typically small range $|R| = \mathcal{O}(n)$
  $\hookrightarrow$ cannot be collision resistant
- should **behave like** $h \sim \mathcal{U}(R^D)$ **in my application**
- should be **fast** to evaluate
- adversarial settings rarely considered

What is a Hash Function?

(it depends on who you ask)

Cryptographic Hash Function

A **collision resistant** function such as $h = \texttt{sha256sum}$

```
$ sha256sum myfile.txt
018a7eaee8a...3e79043e21ab4  myfile.txt
```

Range $R = \{0,1\}^{256}$. It is hard to find x, y with $h(x) = h(y)$.
$\hookrightarrow$ Files with equal hashes are likely the same.

Cryptographic Pseudorandom Function

A function f : Seeds $\times\ D \to R$ where $\log_2 |\text{Seeds}|$ is small and no efficient algorithm can distinguish

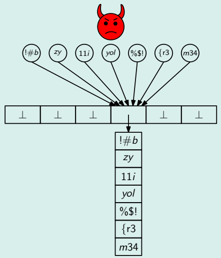
- $f(s, \cdot)$ for $s \sim \mathcal{U}(\text{Seeds})$ and
- $h(\cdot)$ for $h \sim \mathcal{U}(R^D)$,

except with negligible probability.

Hash Function in Algorithm Engineering

- typically small range $|R| = \mathcal{O}(n)$
  $\hookrightarrow$ cannot be collision resistant
- should **behave like** $h \sim \mathcal{U}(R^D)$ **in my application**
- should be **fast** to evaluate
- adversarial settings rarely considered, although:

⚠ HashDoS is a thing.



What is a Hash Function?

(it depends on who you ask)

Cryptographic Hash Function

A **collision resistant** function such as $h = \texttt{sha256sum}$

```
$ sha256sum myfile.txt
018a7eaee8a...3e79043e21ab4  myfile.txt
```

Range $R = \{0,1\}^{256}$. It is hard to find x, y with $h(x) = h(y)$.
$\hookrightarrow$ Files with equal hashes are likely the same.

Cryptographic Pseudorandom Function

A function $f : \text{Seeds} \times D \to R$ where $\log_2 |\text{Seeds}|$ is small and no efficient algorithm can distinguish

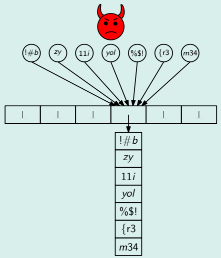
- $f(s, \cdot)$ for $s \sim \mathcal{U}(\text{Seeds})$ and
- $h(\cdot)$ for $h \sim \mathcal{U}(R^D)$,

except with negligible probability.

Hash Function in Algorithm Engineering

- typically small range $|R| = \mathcal{O}(n)$
  $\hookrightarrow$ cannot be collision resistant
- should **behave like** $h \sim \mathcal{U}(R^D)$ **in my application**
- should be **fast** to evaluate
- adversarial settings rarely considered, although:

⚠ HashDoS is a thing.
However: Hash function and hash values need not be public.



Hashing in Practice

Black Magic, do not touch!

MurmurHash

Bitshifts, Magic Constants, . . .

```
uint32_t murmur3_32(const uint8_t* key,
            size_t len, uint32_t seed) {
    uint32_t h = seed;
    uint32_t k;
    for (size_t i = len >> 2; i; i--) {
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    [...]
    return h;
}
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
```

Hashing in Practice

Black Magic, do not touch!

MurmurHash

Bitshifts, Magic Constants, ...

```
uint32_t murmur3_32(const uint8_t* key,
            size_t len, uint32_t seed) {
    uint32_t h = seed;
    uint32_t k;
    for (size_t i = len >> 2; i; i--) {
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    [...]
    return h;
}
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
```

Usage

For $R = [m]$, pick seed $\sim \mathcal{U}(\{0,1\}^{32})$ and use

$$h(x) = \texttt{murmur3\_32}(x, \text{seed}) \bmod m.$$

Hashing in Practice

Black Magic, do not touch!

MurmurHash

Bitshifts, Magic Constants, . . .

```
uint32_t murmur3_32(const uint8_t* key,
            size_t len, uint32_t seed) {
    uint32_t h = seed;
    uint32_t k;
    for (size_t i = len >> 2; i; i--) {
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    [...]
    return h;
}
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
```

Usage

For $R = [m]$, pick seed $\sim \mathcal{U}(\{0,1\}^{32})$ and use

$$h(x) = \texttt{murmur3\_32}(x, \text{seed}) \bmod m.$$

(should avoid modulo in practice, see `https://github.com/lemire/fastrange`)

Hashing in Practice

Black Magic, do not touch!

Karlsruhe Institute of Technology

MurmurHash

Bitshifts, Magic Constants, . . .

```
uint32_t murmur3_32(const uint8_t* key,
            size_t len, uint32_t seed) {
    uint32_t h = seed;
    uint32_t k;
    for (size_t i = len >> 2; i; i--) {
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    [...]
    return h;
}
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
```

Usage

For $R = [m]$, pick seed $\sim \mathcal{U}(\{0,1\}^{32})$ and use

$$h(x) = \texttt{murmur3\_32}(x, \text{seed}) \bmod m.$$

(should avoid modulo in practice, see `https://github.com/lemire/fastrange`)

Does *h* behave like a random function?

Hashing in Practice

Black Magic, do not touch!

MurmurHash

Bitshifts, Magic Constants, . . .

```
uint32_t murmur3_32(const uint8_t* key,
            size_t len, uint32_t seed) {
    uint32_t h = seed;
    uint32_t k;
    for (size_t i = len >> 2; i; i--) {
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    [...]
    return h;
}
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
```

Usage

For $R = [m]$, pick seed $\sim \mathcal{U}(\{0,1\}^{32})$ and use

$$h(x) = \texttt{murmur3\_32}(x, \text{seed}) \bmod m.$$

(should avoid modulo in practice, see `https://github.com/lemire/fastrange`)

Does h behave like a random function?

- **YES,** with respect to many statistical tests.
  see `https://github.com/aappleby/smhasher`

Hashing in Practice

Black Magic, do not touch!

Karlsruhe Institute of Technology

MurmurHash

Bitshifts, Magic Constants, . . .

```
uint32_t murmur3_32(const uint8_t* key,
          size_t len, uint32_t seed) {
    uint32_t h = seed;
    uint32_t k;
    for (size_t i = len >> 2; i; i--) {
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    [...]
    return h;
}
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
```

Usage

For $R = [m]$, pick seed $\sim \mathcal{U}(\{0,1\}^{32})$ and use

$$h(x) = \texttt{murmur3\_32}(x, \mathrm{seed}) \bmod m.$$

(should avoid modulo in practice, see `https://github.com/lemire/fastrange`)

Does h behave like a random function?

- **YES,** with respect to many statistical tests.
  see `https://github.com/aappleby/smhasher`
- **NO,** HashDoS attacks are known.
  see `https://en.wikipedia.org/wiki/MurmurHash#Vulnerabilities`

Hashing in Practice

Black Magic, do not touch!

Karlsruhe Institute of Technology

MurmurHash

Bitshifts, Magic Constants, . . .

```
uint32_t murmur3_32(const uint8_t* key,
            size_t len, uint32_t seed) {
    uint32_t h = seed;
    uint32_t k;
    for (size_t i = len >> 2; i; i--) {
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    [...]
    return h;
}
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
```

Usage

For $R = [m]$, pick seed $\sim \mathcal{U}(\{0,1\}^{32})$ and use

$$h(x) = \texttt{murmur3\_32}(x, \text{seed}) \bmod m.$$

(should avoid modulo in practice, see `https://github.com/lemire/fastrange`)

Does *h* behave like a random function?

- **YES,** with respect to many statistical tests.
  see `https://github.com/aappleby/smhasher`
- **NO,** HashDoS attacks are known.
  see `https://en.wikipedia.org/wiki/MurmurHash#Vulnerabilities`
- **MAYBE,** for your favourite application.

Content

What should a Theorist do?

Approach 1: Ignore the Problem

Simple Uniform Hashing Assumption (SUHA)

- We have access to $h \sim \mathcal{U}(R^D)$ for any R and D.
- h takes $\mathcal{O}(1)$ time to evaluate.
- h takes no space to store.

What should a Theorist do?

Approach 1: Ignore the Problem

Simple Uniform Hashing Assumption (SUHA)

- We have access to $h \sim \mathcal{U}(R^D)$ for any R and D.
- h takes $\mathcal{O}(1)$ time to evaluate.
- h takes no space to store.

How to Analyse your Algorithm

1. *Assume* SUHA holds.
2. *Analyse* algorithm under SUHA.
3. *Hope* that algorithm still works with real hash functions.

What should a Theorist do?

Approach 1: Ignore the Problem

Simple Uniform Hashing Assumption (SUHA)

- We have access to $h \sim \mathcal{U}(R^D)$ for any R and D.
- h takes $\mathcal{O}(1)$ time to evaluate.
- h takes no space to store.

How to Analyse your Algorithm

1. *Assume* SUHA holds.
2. *Analyse* algorithm under SUHA.
3. *Hope* that algorithm still works with real hash functions.

SUHA is "wrong" but adequate

- *Modelling* assumption.
  $\hookrightarrow$ like e.g. ideal gas law in physics

What should a Theorist do?

Approach 1: Ignore the Problem

Karlsruhe Institute of Technology

Simple Uniform Hashing Assumption (SUHA)

- We have access to $h \sim \mathcal{U}(R^D)$ for any R and D.
- h takes $\mathcal{O}(1)$ time to evaluate.
- h takes no space to store.

How to Analyse your Algorithm

1. *Assume* SUHA holds.
2. *Analyse* algorithm under SUHA.
3. *Hope* that algorithm still works with real hash functions.

SUHA is “wrong” but adequate

- *Modelling* assumption.
  $\hookrightarrow$ like e.g. ideal gas law in physics
- Excellent track record in non-adversarial settings.

What should a Theorist do?
Approach 2: Bring your own Hash Functions

Karlsruhe Institute of Technology

Analyse Algorithm using Universal Hashing

1. Define family $\mathcal{H} \subseteq R^D$ of hash functions with $\log(|\mathcal{H}|)$ not too large.
   $\hookrightarrow$ sampling and storing $h \in \mathcal{H}$ is cheap
2. Proof that algorithm with $h \sim \mathcal{U}(\mathcal{H})$ has good expected behaviour.

What should a Theorist do?

Approach 2: Bring your own Hash Functions

Analyse Algorithm using Universal Hashing

1. Define family $\mathcal{H} \subseteq R^D$ of hash functions with $\log(|\mathcal{H}|)$ not too large.
   $\hookrightarrow$ sampling and storing $h \in \mathcal{H}$ is cheap
2. Proof that algorithm with $h \sim \mathcal{U}(\mathcal{H})$ has good expected behaviour.

Remarks

What should a Theorist do?

Approach 2: Bring your own Hash Functions

Analyse Algorithm using Universal Hashing

1. Define family $\mathcal{H} \subseteq R^D$ of hash functions with $\log(|\mathcal{H}|)$ not too large.
   $\hookrightarrow$ sampling and storing $h \in \mathcal{H}$ is cheap
2. Proof that algorithm with $h \sim \mathcal{U}(\mathcal{H})$ has good expected behaviour.

Remarks

- Mathematical structure of $\mathcal{H}$ must be amenable to analysis.

What should a Theorist do?
Approach 2: Bring your own Hash Functions

Analyse Algorithm using Universal Hashing

1. Define family $\mathcal{H} \subseteq R^D$ of hash functions with $\log(|\mathcal{H}|)$ not too large.
   $\hookrightarrow$ sampling and storing $h \in \mathcal{H}$ is cheap
2. Proof that algorithm with $h \sim \mathcal{U}(\mathcal{H})$ has good expected behaviour.

Remarks

- Mathematical structure of $\mathcal{H}$ must be amenable to analysis.
- *Rigorously* covers non-adversarial settings.

What should a Theorist do?

Approach 2: Bring your own Hash Functions

Analyse Algorithm using Universal Hashing

1. Define family $\mathcal{H} \subseteq R^D$ of hash functions with $\log(|\mathcal{H}|)$ not too large.
   $\hookrightarrow$ sampling and storing $h \in \mathcal{H}$ is cheap
2. Proof that algorithm with $h \sim \mathcal{U}(\mathcal{H})$ has good expected behaviour.

Remarks

- Mathematical structure of $\mathcal{H}$ must be amenable to analysis.
- *Rigorously* covers non-adversarial settings.
- Proofs often difficult.
  $\hookrightarrow$ Wider theory practice gap than with SUHA.

What should a Theorist do?

Approach 3: Let the Cryptographers do the Work

Karlsruhe Institute of Technology

How to Analyse your Algorithm using Cryptographic Assumptions

1. Analyse algorithm under *SUHA*.
2. Actually use *cryptographic pseudorandom function* f.
   - **Case 1:** Everything still works. Great! :-)
   - **Case 2:** Something fails.

     $\Rightarrow$ Your use case can tell the difference between f and true randomness.

     $\hookrightarrow$ The cryptographers said this is impossible. $\lightning$

What should a Theorist do?

Approach 3: Let the Cryptographers do the Work

Karlsruhe Institute of Technology

How to Analyse your Algorithm using Cryptographic Assumptions

1. Analyse algorithm under *SUHA*.
2. Actually use *cryptographic pseudorandom function* f.
   - **Case 1:** Everything still works. Great! :-)
   - **Case 2:** Something fails.

     $\Rightarrow$ Your use case can tell the difference between f and true randomness.

     $\hookrightarrow$ The cryptographers said this is impossible. $\lightning$

Should we use cryptographic pseudorandom functions?

What should a Theorist do?

Approach 3: Let the Cryptographers do the Work

Karlsruhe Institute of Technology

How to Analyse your Algorithm using Cryptographic Assumptions

1. Analyse algorithm under *SUHA*.
2. Actually use *cryptographic pseudorandom function* f.
   - **Case 1:** Everything still works. Great! :-)
   - **Case 2:** Something fails.
     $\Rightarrow$ Your use case can tell the difference between f and true randomness.
     $\hookrightarrow$ The cryptographers said this is impossible. ↯

Should we use cryptographic pseudorandom functions?

- **YES.** Algorithms become robust even in some adversarial settings.
  $\hookrightarrow$ e.g. Python, Haskell, Ruby, Rust use **SipHash** by default
  `https://en.wikipedia.org/wiki/SipHash`

What should a Theorist do?

Approach 3: Let the Cryptographers do the Work

Karlsruhe Institute of Technology

How to Analyse your Algorithm using Cryptographic Assumptions

1. Analyse algorithm under *SUHA*.
2. Actually use *cryptographic pseudorandom function* f.
   - **Case 1:** Everything still works. Great! :-)
   - **Case 2:** Something fails.

     $\Rightarrow$ Your use case can tell the difference between f and true randomness.

     $\hookrightarrow$ The cryptographers said this is impossible. $\lightning$

Should we use cryptographic pseudorandom functions?

- **YES.** Algorithms become robust even in some adversarial settings.

  $\hookrightarrow$ e.g. Python, Haskell, Ruby, Rust use **SipHash** by default

  `https://en.wikipedia.org/wiki/SipHash`
- **NO.** Too slow in high-performance settings.

Hash Function	MiB / sec
SipHash	944
Murmur3F	7623
xxHash64	12109

(source: `https://github.com/rurban/smhasher`)

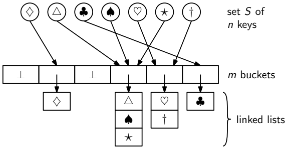
Content

Karlsruhe Institute of Technology

Hash Table with Chaining

Search Time under Chaining

$$\max_{\substack{S \subseteq D \\ |S|=n}} \max_{x \in D} \quad 1 + |\{y \in S \mid h(y) = h(x)\}|$$

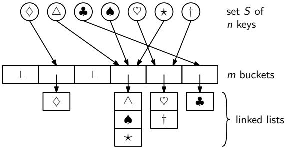


Hash Table with Chaining

Search Time under Chaining

For $n, m \in \mathbb{N}$ and a family $\mathcal{H} \subseteq [m]^D$ of hash functions the *maximum expected search time* is at most

$$T_{\text{chaining}}(n, m, \mathcal{H}) = \max_{\substack{S \subseteq D \\ |S| = n}} \max_{x \in D} \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big]$$



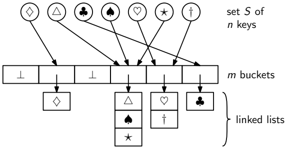
Hash Table with Chaining

Search Time under Chaining

For $n, m \in \mathbb{N}$ and a family $\mathcal{H} \subseteq [m]^D$ of hash functions the *maximum expected search time* is at most

$$T_{\text{chaining}}(n, m, \mathcal{H}) = \max_{\substack{S \subseteq D \\ |S| = n}} \max_{x \in D} \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})} \Big[1 + |\{y \in S \mid h(y) = h(x)\}| \Big]$$

Key set is *worst case*. Only $h \in \mathcal{H}$ is random. Key set is fixed *before* h is chosen.



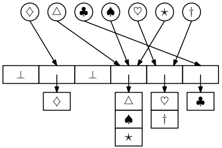
Hash Table with Chaining

Search Time under Chaining

For $n, m \in \mathbb{N}$ and a family $\mathcal{H} \subseteq [m]^D$ of hash functions the *maximum expected search time* is at most

$$T_{\text{chaining}}(n, m, \mathcal{H}) = \max_{\substack{S \subseteq D \\ |S|=n}} \max_{x \in D} \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big]$$

⚠ Key set is *worst case*. Only $h \in \mathcal{H}$ is random. Key set is fixed *before* h is chosen.



Theorem: Hash Table with Chaining under SUHA

If $\mathcal{H} = [m]^D$ then $T_{\text{chaining}}(n, m, \mathcal{H}) \leq 2 + \alpha = \mathcal{O}(1)$ if $\alpha \in \mathcal{O}(1)$.

Analysis of Hash Table with Chaining under SUHA

Karlsruhe Institute of Technology

Theorem: Hash Table with Chaining under SUHA

Let $\mathcal{H} = [m]^D$, $S \subseteq D$ with $|S| = n$ and $x \in D$ then

$$\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\left[1 + |\{y \in S \mid h(y) = h(x)\}|\right] \leq 2 + \alpha$$

Analysis of Hash Table with Chaining under SUHA

Karlsruhe Institute of Technology

Theorem: Hash Table with Chaining under SUHA

Let $\mathcal{H} = [m]^D$, $S \subseteq D$ with $|S| = n$ and $x \in D$ then

$$\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \leq 2 + \alpha$$

Proof.

$$\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big]$$

Analysis of Hash Table with Chaining under SUHA

Karlsruhe Institute of Technology

Theorem: Hash Table with Chaining under SUHA

Let $\mathcal{H} = [m]^D$, $S \subseteq D$ with $|S| = n$ and $x \in D$ then

$$\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \leq 2 + \alpha$$

Proof.

$$\begin{aligned} &\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \\ = &\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + \sum_{y \in S} \mathbb{1}_{\{h(y)=h(x)\}}\Big] \end{aligned}$$

Analysis of Hash Table with Chaining under SUHA

Karlsruhe Institute of Technology

Theorem: Hash Table with Chaining under SUHA

Let $\mathcal{H} = [m]^D$, $S \subseteq D$ with $|S| = n$ and $x \in D$ then

$$\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \leq 2 + \alpha$$

Proof.

$$\begin{aligned}
&\mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \\
&= \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + \sum_{y \in S} \mathbb{1}_{\{h(y)=h(x)\}}\Big] \\
&= 1 + \sum_{y \in S} \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[\mathbb{1}_{\{h(y)=h(x)\}}\Big]
\end{aligned}$$

Analysis of Hash Table with Chaining under SUHA

Karlsruhe Institute of Technology

Theorem: Hash Table with Chaining under SUHA

Let $\mathcal{H} = [m]^D$, $S \subseteq D$ with $|S| = n$ and $x \in D$ then

$$\mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \leq 2 + \alpha$$

Proof.

$$\begin{aligned}
&\mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \\
&= \mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + \sum_{y\in S} \mathbb{1}_{\{h(y)=h(x)\}}\Big] \\
&= 1 + \sum_{y\in S} \mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[\mathbb{1}_{\{h(y)=h(x)\}}\Big]
\end{aligned}$$

$$= 1 + \sum_{y\in S} \Pr_{h\sim\mathcal{U}(\mathcal{H})}[h(y) = h(x)]$$

Analysis of Hash Table with Chaining under SUHA

Karlsruhe Institute of Technology

Theorem: Hash Table with Chaining under SUHA

Let $\mathcal{H} = [m]^D$, $S \subseteq D$ with $|S| = n$ and $x \in D$ then

$$\mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \leq 2 + \alpha$$

Proof.

$$\begin{aligned}
&\mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \\
&= \mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + \sum_{y\in S} \mathbb{1}_{\{h(y)=h(x)\}}\Big] \\
&= 1 + \sum_{y\in S} \mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[\mathbb{1}_{\{h(y)=h(x)\}}\Big]
\end{aligned}$$

$$\begin{aligned}
&= 1 + \sum_{y\in S} \Pr_{h\sim\mathcal{U}(\mathcal{H})}[h(y) = h(x)] \\
&= 1 + 1 + \sum_{y\in S\setminus\{x\}} \Pr_{h\sim\mathcal{U}(\mathcal{H})}[h(y) = h(x)]
\end{aligned}$$

Analysis of Hash Table with Chaining under SUHA

Karlsruhe Institute of Technology

Theorem: Hash Table with Chaining under SUHA

Let $\mathcal{H} = [m]^D$, $S \subseteq D$ with $|S| = n$ and $x \in D$ then

$$\mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \leq 2 + \alpha$$

Proof.

$$\begin{aligned}
&\mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big] \\
&= \mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[1 + \sum_{y\in S} \mathbb{1}_{\{h(y)=h(x)\}}\Big] \\
&= 1 + \sum_{y\in S} \mathbb{E}_{h\sim\mathcal{U}(\mathcal{H})}\Big[\mathbb{1}_{\{h(y)=h(x)\}}\Big] \\
&= 1 + \sum_{y\in S} \Pr_{h\sim\mathcal{U}(\mathcal{H})}[h(y) = h(x)] \\
&= 1 + 1 + \sum_{y\in S\setminus\{x\}} \Pr_{h\sim\mathcal{U}(\mathcal{H})}[h(y) = h(x)] \\
&= 2 + \sum_{y\in S\setminus\{x\}} \frac{1}{m} \leq 2 + \frac{n}{m} = 2 + \alpha. \quad \square
\end{aligned}$$

Content

A Universal Hash Family

Karlsruhe Institute of Technology

Definition: c-universal hash family

A class $\mathcal{H} \subseteq [m]^D$ is called *c-universal* if: $\forall x \neq y \in D : \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x) = h(y)] \leq \frac{c}{m}$.

A Universal Hash Family

Karlsruhe Institute of Technology

Definition: c-universal hash family

A class $\mathcal{H} \subseteq [m]^D$ is called *c-universal* if: $\quad \forall x \neq y \in D : \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x) = h(y)] \leq \frac{c}{m}$.

Note: $\mathcal{H} = [m]^D$ is 1-universal.

A Universal Hash Family

Karlsruhe Institute of Technology

Definition: c-universal hash family

A class $\mathcal{H} \subseteq [m]^D$ is called *c-universal* if: $\forall x \neq y \in D : \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x) = h(y)] \leq \frac{c}{m}$.

A Universal Hash Family

Definition: c-universal hash family

A class $\mathcal{H} \subseteq [m]^D$ is called *c-universal* if: $\forall x \neq y \in D : \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x) = h(y)] \leq \frac{c}{m}$.

Reminder (?): Finite Fields

Let $\mathbb{F}_p = \{0, \ldots, p-1\}$ for a prime number p. Then $(\mathbb{F}_p, \times, \oplus)$ is a field where

$$a \times b := (a \cdot b) \bmod p \qquad \textit{and} \qquad a \oplus b := (a + b) \bmod p.$$

In particular $(\mathbb{F}_p^* := \mathbb{F}_p \setminus \{0\}, \times)$ is a group.

A Universal Hash Family

Karlsruhe Institute of Technology

Definition: c-universal hash family

A class $\mathcal{H} \subseteq [m]^D$ is called *c-universal* if: $\quad \forall x \neq y \in D : \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x) = h(y)] \leq \frac{c}{m}$.

Reminder (?): Finite Fields

Let $\mathbb{F}_p = \{0, \ldots, p-1\}$ for a prime number p. Then $(\mathbb{F}_p, \times, \oplus)$ is a field where

$$a \times b := (a \cdot b) \bmod p \qquad \textit{and} \qquad a \oplus b := (a + b) \bmod p.$$

In particular $(\mathbb{F}_p^* := \mathbb{F}_p \setminus \{0\}, \times)$ is a group.

The class of Linear Hash Functions

Assume $D \subseteq \mathbb{F}_p$ for prime p. Then the following class is 1-universal:

$$\mathcal{H}_{p,m}^{\text{lin}} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}.$$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned}$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

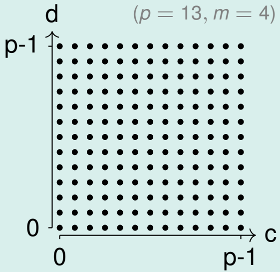
Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{array}{l} c = (a \times x) \oplus b \\ d = (a \times y) \oplus b \end{array} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{matrix} c = (a \times x) \oplus b \\ d = (a \times y) \oplus b \end{matrix} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p \times \mathbb{F}_p \to \mathbb{F}_p \times \mathbb{F}_p$

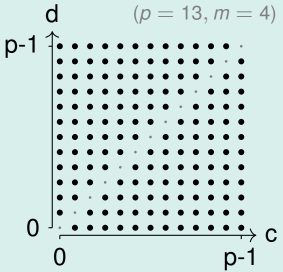


Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{matrix} c = (a \times x) \oplus b \\ d = (a \times y) \oplus b \end{matrix} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^{*} \times \mathbb{F}_p \to \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

$(p = 13, m = 4)$

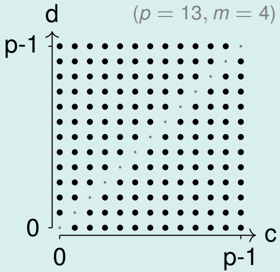


$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^* \times \mathbb{F}_p \to P$.

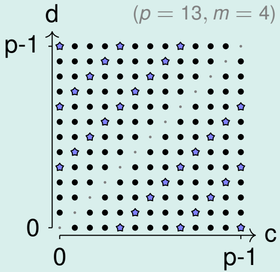


$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}$.
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^* \times \mathbb{F}_p \to P$.
- Define *bad set* $B := \{(c, d) \in P \mid c \bmod m = d \bmod m\}$.
  $\hookrightarrow$ from picture: $\frac{|B|}{|P|} \leq \frac{1}{m}$.

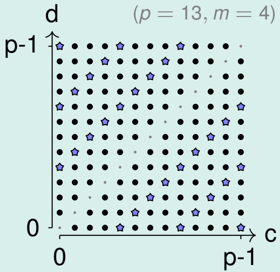


$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}$.
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^* \times \mathbb{F}_p \to P$.
- Define *bad set* $B := \{(c, d) \in P \mid c \bmod m = d \bmod m\}$.
  $\hookrightarrow$ from picture: $\frac{|B|}{|P|} \leq \frac{1}{m}$.



$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

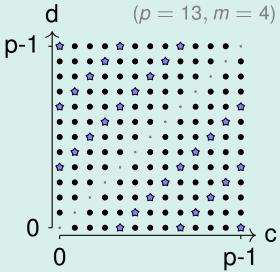
$$\Pr_{a,b \sim \mathcal{U}(\mathbb{F}_p^* \times \mathbb{F}_p)}[h(x) = h(y)]$$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^* \times \mathbb{F}_p \to P$.
- Define *bad set* $B := \{(c, d) \in P \mid c \bmod m = d \bmod m\}$.

  $\hookrightarrow$ from picture: $\frac{|B|}{|P|} \leq \frac{1}{m}$.



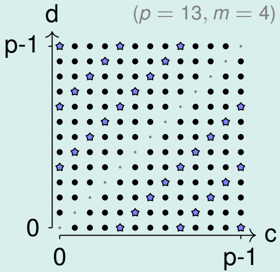
$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

$$\Pr_{a,b \sim \mathcal{U}(\mathbb{F}_p^* \times \mathbb{F}_p)}[h(x) = h(y)] = \Pr_{a,b}[((a \times x) \oplus b) \bmod m = ((a \times y) \oplus b) \bmod m]$$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}^*_p, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}$.
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}^*_p \times \mathbb{F}_p \to P$.
- Define *bad set* $B := \{(c, d) \in P \mid c \bmod m = d \bmod m\}$.
  $\hookrightarrow$ from picture: $\frac{|B|}{|P|} \leq \frac{1}{m}$.



$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

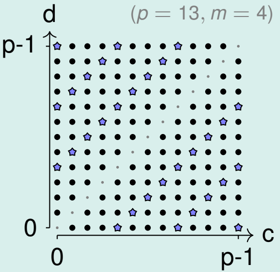
$$\Pr_{a,b \sim \mathcal{U}(\mathbb{F}^*_p \times \mathbb{F}_p)}[h(x) = h(y)] = \Pr_{a,b}[((a \times x) \oplus b) \bmod m = ((a \times y) \oplus b) \bmod m]$$

$$= \Pr_{a,b}[c \bmod m = d \bmod m] = \Pr_{a,b}[(c, d) \in B]$$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{array}{l} c = (a \times x) \oplus b \\ d = (a \times y) \oplus b \end{array} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^* \times \mathbb{F}_p \to P$.
- Define *bad set* $B := \{(c, d) \in P \mid c \bmod m = d \bmod m\}$.
  $\hookrightarrow$ from picture: $\frac{|B|}{|P|} \leq \frac{1}{m}$.



$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

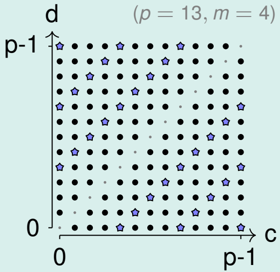
$$\Pr_{a,b \sim \mathcal{U}(\mathbb{F}_p^* \times \mathbb{F}_p)}[h(x) = h(y)] = \Pr_{a,b}[((a \times x) \oplus b) \bmod m = ((a \times y) \oplus b) \bmod m]$$

$$= \Pr_{a,b}[c \bmod m = d \bmod m] = \Pr_{a,b}[(c, d) \in B] = \Pr_{c,d \sim \mathcal{U}(P)}[(c, d) \in B]$$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h\sim\mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^* \times \mathbb{F}_p \to P$.
- Define *bad set* $B := \{(c, d) \in P \mid c \bmod m = d \bmod m\}$.
  $\hookrightarrow$ from picture: $\frac{|B|}{|P|} \leq \frac{1}{m}$.



$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

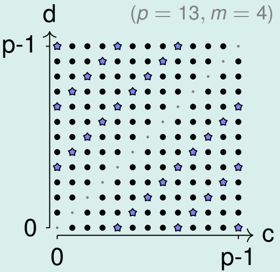
$$\Pr_{a,b\sim\mathcal{U}(\mathbb{F}_p^* \times \mathbb{F}_p)}[h(x) = h(y)] = \Pr_{a,b}[((a \times x) \oplus b) \bmod m = ((a \times y) \oplus b) \bmod m]$$

$$= \Pr_{a,b}[c \bmod m = d \bmod m] = \Pr_{a,b}[(c, d) \in B] = \Pr_{c,d\sim\mathcal{U}(P)}[(c, d) \in B] = \frac{|B|}{|P|}$$

Proof that $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times x) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$ is 1-universal.

Let $x \neq y \in \mathbb{F}_p$. (To show: $\Pr_{h \sim \mathcal{H}^{\text{lin}}_{p,m}}[h(x) = h(y)] \leq 1/m$.)

- Define $\begin{aligned} c &= (a \times x) \oplus b \\ d &= (a \times y) \oplus b \end{aligned} \Leftrightarrow \begin{pmatrix} c \\ d \end{pmatrix} = \underbrace{\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}}_{\text{regular!}} \begin{pmatrix} a \\ b \end{pmatrix}.$
- The mapping $(a, b) \mapsto (c, d)$ is a bijection (for every $x \neq y$) from $\mathbb{F}_p^* \times \mathbb{F}_p \to P$.
- Define *bad set* $B := \{(c, d) \in P \mid c \bmod m = d \bmod m\}$.
  $\hookrightarrow$ from picture: $\frac{|B|}{|P|} \leq \frac{1}{m}$.



$P := \mathbb{F}_p \times \mathbb{F}_p \setminus \{(b, b) \mid b \in \mathbb{F}_p\}$

$$\Pr_{a,b \sim \mathcal{U}(\mathbb{F}_p^* \times \mathbb{F}_p)}[h(x) = h(y)] = \Pr_{a,b}[((a \times x) \oplus b) \bmod m = ((a \times y) \oplus b) \bmod m]$$

$$= \Pr_{a,b}[c \bmod m = d \bmod m] = \Pr_{a,b}[(c, d) \in B] = \Pr_{c,d \sim \mathcal{U}(P)}[(c, d) \in B] = \frac{|B|}{|P|} \leq \frac{1}{m}. \quad \square$$

Analysis of Hash Table with Chaining

… using a Universal Hash Family

Theorem

If $\mathcal{H} \subseteq [m]^D$ is a c-universal hash family then $T_{\text{chaining}}(n, m, \mathcal{H}) \leq 2 + c\alpha = \mathcal{O}(1)$ if $\alpha \in \mathcal{O}(1)$ and $c \in \mathcal{O}(1)$.

Proof: Mostly the same.

$$\forall S \subseteq [D], \forall x \in D : \qquad \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big]$$

Analysis of Hash Table with Chaining

. . . using a Universal Hash Family

Theorem

If $\mathcal{H} \subseteq [m]^D$ is a c-universal hash family then $T_{\text{chaining}}(n, m, \mathcal{H}) \leq 2 + c\alpha = \mathcal{O}(1)$ if $\alpha \in \mathcal{O}(1)$ and $c \in \mathcal{O}(1)$.

Proof: Mostly the same.

$$\forall S \subseteq [D], \forall x \in D: \qquad \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big]$$

$$= \ldots = 2 + \sum_{y \in S \setminus \{x\}} \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(y) = h(x)]$$

Analysis of Hash Table with Chaining

... using a Universal Hash Family

Karlsruhe Institute of Technology

Theorem

If $\mathcal{H} \subseteq [m]^D$ is a c-universal hash family then $T_{\text{chaining}}(n, m, \mathcal{H}) \leq 2 + c\alpha = \mathcal{O}(1)$ if $\alpha \in \mathcal{O}(1)$ and $c \in \mathcal{O}(1)$.

Proof: Mostly the same.

$$\forall S \subseteq [D], \forall x \in D : \quad \mathbb{E}_{h \sim \mathcal{U}(\mathcal{H})}\Big[1 + |\{y \in S \mid h(y) = h(x)\}|\Big]$$

$$= \ldots = 2 + \sum_{y \in S \setminus \{x\}} \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(y) = h(x)]$$

$$= 2 + \sum_{y \in S \setminus \{x\}} \frac{c}{m} \leq 2 + \frac{cn}{m} = 2 + c\alpha. \quad \square$$

More Universal Families

Examples for Universal Hash Families

- "$\boldsymbol{((ax+b) \bmod p) \bmod m}$" is 1-universal

as discussed: $D = \mathbb{F}_p, \qquad R = [m],$

$$\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times b) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$$

More Universal Families

Examples for Universal Hash Families

- “$((\boldsymbol{ax} + \boldsymbol{b})$ **mod** $\boldsymbol{p})$ **mod** $\boldsymbol{m}$” is 1-universal

  $$\text{as discussed: } D = \mathbb{F}_p, \qquad R = [m],$$
  $$\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times b) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$$

- “$(\boldsymbol{ax}$ **mod** $\boldsymbol{p})$ **mod** $\boldsymbol{m}$” is only 2-universal:

  $$D = \mathbb{F}_p, \qquad R = [m],$$
  $$\mathcal{H} = \{x \mapsto (a \times b) \bmod m \mid a \in \mathbb{F}_p^*\}$$

More Universal Families

Karlsruhe Institute of Technology

Examples for Universal Hash Families

- "$((\boldsymbol{ax} + \boldsymbol{b})$ **mod** $\boldsymbol{p})$ **mod** $\boldsymbol{m}$" is 1-universal

  as discussed: $D = \mathbb{F}_p, \qquad R = [m],$

  $$\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times b) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$$

- "$(\boldsymbol{ax}$ **mod** $\boldsymbol{p})$ **mod** $\boldsymbol{m}$" is only 2-universal:

  $$D = \mathbb{F}_p, \qquad R = [m],$$

  $$\mathcal{H} = \{x \mapsto (a \times b) \bmod m \mid a \in \mathbb{F}_p^*\}$$

- **Multiply-Shift** is 2-universal:

  $$D = \{0, \dots, 2^w - 1\}, \qquad R = \{0, \dots, 2^\ell - 1\}$$

  $$\mathcal{H} = \{x \mapsto \lfloor ((a \cdot x + b) \bmod 2^w)/2^{w-\ell} \rfloor \mid$$
  $$\textbf{odd } a \in \{1, \dots, 2^w - 1\}, b \in \{0, \dots, 2^w - 1\}.\}$$

More Universal Families

Karlsruhe Institute of Technology

Examples for Universal Hash Families

- “$((ax + b) \bmod p) \bmod m$” is 1-universal

  as discussed: $D = \mathbb{F}_p$, $\quad R = [m]$,

  $\mathcal{H}^{\text{lin}}_{p,m} := \{x \mapsto ((a \times b) \oplus b) \bmod m \mid a \in \mathbb{F}_p^*, b \in \mathbb{F}_p\}$

- “$(ax \bmod p) \bmod m$” is only 2-universal:

  $D = \mathbb{F}_p$, $\quad R = [m]$,

  $\mathcal{H} = \{x \mapsto (a \times b) \bmod m \mid a \in \mathbb{F}_p^*\}$

- **Multiply-Shift** is 2-universal:

  $D = \{0, \ldots, 2^w - 1\}$, $\quad R = \{0, \ldots, 2^\ell - 1\}$

  $\mathcal{H} = \{x \mapsto \lfloor ((a \cdot x + b) \bmod 2^w)/2^{w-\ell} \rfloor \mid$

  odd $a \in \{1, \ldots, 2^w - 1\}, b \in \{0, \ldots, 2^w - 1\}.\}$

Selling point of multiply shift:

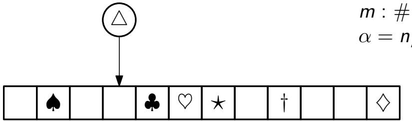
- “$x \bmod 2^w$” drops some higher order bits
- “$\lfloor x/2^{w-\ell} \rfloor$ drops some lower order bits
- No division or modulo operation needed!

For $w = 32$ (taken from Thorup 2015):

```
uint32_t hash(uint32_t x, uint32_t l, uint64_t a) {
        return (a * x + b) >> (64-l);
}
```

Content

Hash Table with Linear Probing



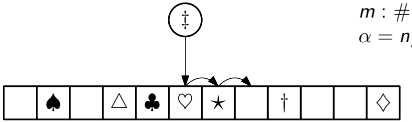
S : set of n keys
m : $\#$ of buckets
$\alpha = n/m$

Hash Table with Linear Probing

S : set of n keys
m : $\#$ of buckets
$\alpha = n/m$

	$\spadesuit$		$\triangle$	$\clubsuit$	$\heartsuit$	$\star$		$\dagger$			$\diamondsuit$

Hash Table with Linear Probing



S : set of n keys
m : $\#$ of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \dots \pmod m$.

Insert. Put x into first empty bucket.

Hash Table with Linear Probing

S : set of n keys
m : # of buckets
$\alpha = n/m$

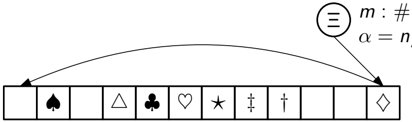
	$\spadesuit$		$\triangle$	$\clubsuit$	$\heartsuit$	$\star$	$\ddagger$	$\dagger$			$\diamondsuit$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod{m}$.

Insert. Put x into first empty bucket.

Hash Table with Linear Probing



Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \dots \pmod{m}$.

Insert. Put x into first empty bucket.

Hash Table with Linear Probing

S : set of n keys
m : # of buckets
$\alpha = n/m$

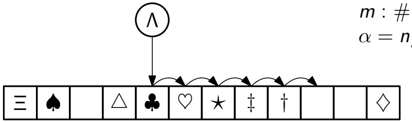
Ξ	$\spadesuit$		$\triangle$	$\clubsuit$	$\heartsuit$	$\star$	$\ddagger$	$\dagger$			$\diamondsuit$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod{m}$.

Insert. Put x into first empty bucket.

Hash Table with Linear Probing



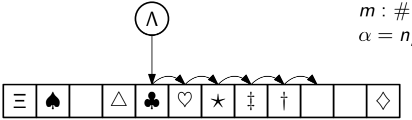
S : set of n keys
m : # of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod m$.

Insert. Put x into first empty bucket.

Hash Table with Linear Probing



S : set of n keys
m : $\#$ of buckets
$\alpha = n/m$

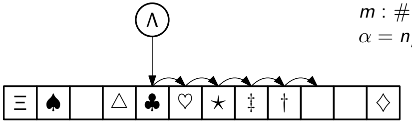
Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod{m}$.

Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

Hash Table with Linear Probing



S : set of n keys
m : # of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \dots \pmod{m}$.

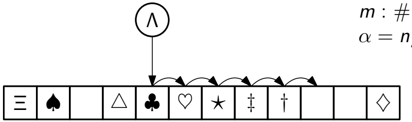
Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

Delete. Lookup and remove x and ⚠ check if a key to the right wants to move into the hole.

↪ For details see `https://en.wikipedia.org/wiki/Linear_probing`

Hash Table with Linear Probing



S : set of n keys
m : # of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod{m}$.

Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

Delete. Lookup and remove x and ⚠ check if a key to the right wants to move into the hole.

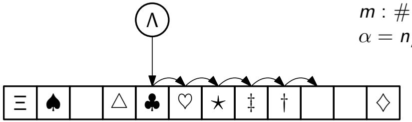
↪ For details see `https://en.wikipedia.org/wiki/Linear_probing`

Running Times

- Lookup($x \in S$): At most x's insertion time.

Hash Table with Linear Probing

Karlsruhe Institute of Technology



S : set of n keys
m : # of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod{m}$.

Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

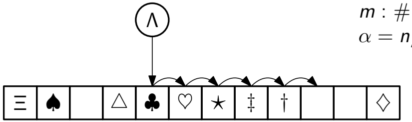
Delete. Lookup and remove x and ⚠ check if a key to the right wants to move into the hole.

↪ For details see `https://en.wikipedia.org/wiki/Linear_probing`

Running Times

- Lookup($x \in S$): At most x's insertion time.
- Lookup($x \notin S$): At most the time it *would take* to insert x now.

Hash Table with Linear Probing



S : set of n keys
m : # of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \dots \pmod{m}$.

Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

Delete. Lookup and remove x and ⚠ check if a key to the right wants to move into the hole.

↪ For details see `https://en.wikipedia.org/wiki/Linear_probing`

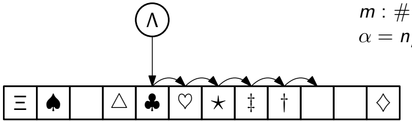
Running Times

- Lookup($x \in S$): At most x's insertion time.
- Lookup($x \notin S$): At most the time it *would take* to insert x now.
- Delete($x \in S$): At most the time it *would take* to insert $y \notin S$ with $h(y) = h(x)$.

Hash Table with Linear Probing

Karlsruhe Institute of Technology

S : set of n keys
m : # of buckets
$\alpha = n/m$



Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod{m}$.

Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

Delete. Lookup and remove x and ⚠ check if a key to the right wants to move into the hole.

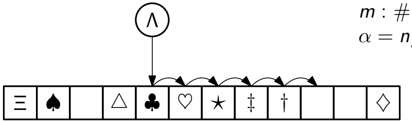
↪ For details see `https://en.wikipedia.org/wiki/Linear_probing`

Running Times

- Lookup($x \in S$): At most x's insertion time.
- Lookup($x \notin S$): At most the time it *would take* to insert x now.
- Delete($x \in S$): At most the time it *would take* to insert $y \notin S$ with $h(y) = h(x)$.

↪ It suffices to understand insertion times!

Hash Table with Linear Probing



S : set of n keys
m : # of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \dots \pmod{m}$.

Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

Delete. Lookup and remove x and ⚠ check if a key to the right wants to move into the hole.

$\hookrightarrow$ For details see `https://en.wikipedia.org/wiki/Linear_probing`

Running Times

- Lookup($x \in S$): At most x's insertion time.
- Lookup($x \notin S$): At most the time it *would take* to insert x now.
- Delete($x \in S$): At most the time it *would take* to insert $y \notin S$ with $h(y) = h(x)$.

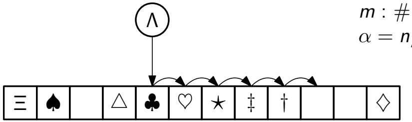
$\hookrightarrow$ It suffices to understand insertion times!

Theorem: Linear Probing under SUHA

Let $T_{n,m}$ be the random insertion time into a linear probing hash table. If $\frac{1}{2} \leq \alpha = \frac{n}{m} < \alpha_{\max}$ for some $\alpha_{\max} < 1$ then under SUHA we have

$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1).$$

Hash Table with Linear Probing



S : set of n keys
m : # of buckets
$\alpha = n/m$

Operations

For key x *probe* buckets $h(x), h(x)+1, h(x)+2, \ldots \pmod{m}$.

Insert. Put x into first empty bucket.

Lookup. Look for x, abort when encountering empty bucket.

Delete. Lookup and remove x and ⚠ check if a key to the right wants to move into the hole.

$\hookrightarrow$ For details see `https://en.wikipedia.org/wiki/Linear_probing`

Running Times

- Lookup($x \in S$): At most x's insertion time.
- Lookup($x \notin S$): At most the time it *would take* to insert x now.
- Delete($x \in S$): At most the time it *would take* to insert $y \notin S$ with $h(y) = h(x)$.

$\hookrightarrow$ It suffices to understand insertion times!

Theorem: Linear Probing under SUHA

Let $T_{n,m}$ be the random insertion time into a linear probing hash table. If $\frac{1}{2} \leq \alpha = \frac{n}{m} < \alpha_{\max}$ for some $\alpha_{\max} < 1$ then under SUHA we have

$$\mathbb{E}[T_{n,m}] = \mathcal{O}\left(\frac{1}{(1-\alpha_{\max})^2}\right) = \mathcal{O}(1). \quad \text{(not here)}$$

Content

Preparation: A concentration bound

Chernoff

For $X \sim \mathit{Bin}(n, p)$ and $\varepsilon \in [0, 1]$ we have $\Pr[X \geq (1 + \varepsilon)\mathbb{E}[X]] \leq \exp(-\varepsilon^2\mathbb{E}[X]/3)$.

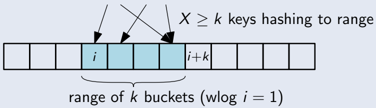
Preparation: A concentration bound

Karlsruhe Institute of Technology

Chernoff

For $X \sim Bin(n,p)$ and $\varepsilon \in [0,1]$ we have $\Pr[X \geq (1+\varepsilon)\mathbb{E}[X]] \leq \exp(-\varepsilon^2 \mathbb{E}[X]/3)$.

Lemma: $\Pr[\geq k$ hits in segment of length $k]$



Let $k \in \mathbb{N}$ and $X = |\{y \in S \mid h(y) \in \{1,\dots,k\}\}|$.

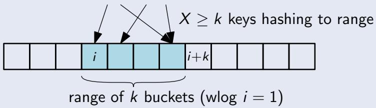
Then $\Pr_{h \sim \mathcal{U}(R^D)}[X \geq k] \leq \exp(-(1-\alpha)^2 k/3)$.

Preparation: A concentration bound

Chernoff

For $X \sim Bin(n, p)$ and $\varepsilon \in [0, 1]$ we have $\Pr[X \geq (1+\varepsilon)\mathbb{E}[X]] \leq \exp(-\varepsilon^2 \mathbb{E}[X]/3)$.

Lemma: $\Pr[\geq k$ hits in segment of length $k]$



Let $k \in \mathbb{N}$ and $X = |\{y \in S \mid h(y) \in \{1, \dots, k\}\}|$.

Then $\Pr_{h \sim \mathcal{U}(R^D)}[X \geq k] \leq \exp(-(1-\alpha)^2 k/3)$.

Proof

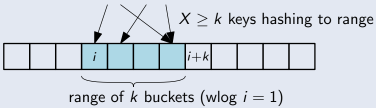
Let $S = \{x_1, \dots, x_n\}$ and $X_i = \mathbb{1}_{\{h(x_i) \in \{1,\dots,k\}\}} \sim Ber(\frac{k}{m})$.
Then $X = \sum_{i \in [n]} X_i \sim Bin(n, \frac{k}{m})$ with $\mathbb{E}[X] = \frac{kn}{m} = \alpha k$.

Preparation: A concentration bound

Chernoff

For $X \sim Bin(n,p)$ and $\varepsilon \in [0,1]$ we have $\Pr[X \geq (1+\varepsilon)\mathbb{E}[X]] \leq \exp(-\varepsilon^2\mathbb{E}[X]/3)$.

Lemma: $\Pr[\geq k$ hits in segment of length $k]$



Let $k \in \mathbb{N}$ and $X = |\{y \in S \mid h(y) \in \{1,\dots,k\}\}|$.

Then $\Pr_{h \sim \mathcal{U}(R^D)}[X \geq k] \leq \exp(-(1-\alpha)^2 k/3)$.

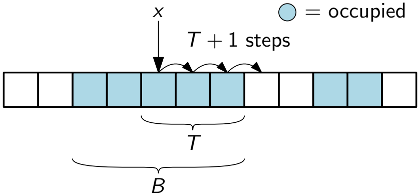
Proof

Let $S = \{x_1,\dots,x_n\}$ and $X_i = \mathbb{1}_{\{h(x_i)\in\{1,\dots,k\}\}} \sim Ber(\frac{k}{m})$.
Then $X = \sum_{i\in[n]} X_i \sim Bin(n, \frac{k}{m})$ with $\mathbb{E}[X] = \frac{kn}{m} = \alpha k$.

$$\begin{aligned}
\Pr[X \geq k] &= \Pr[X \geq \tfrac{1}{\alpha}\mathbb{E}[X]] \\
&= \Pr[X \geq (1 + \tfrac{1-\alpha}{\alpha})\mathbb{E}[X]] \\
&\leq \exp(-(\tfrac{1-\alpha}{\alpha})^2 \alpha k/3) \\
&\leq \exp(-(1-\alpha)^2 k/3). \quad (\text{using } \tfrac{1}{2} \leq \alpha \leq 1)
\end{aligned}$$

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$\mathbb{E}[T]$

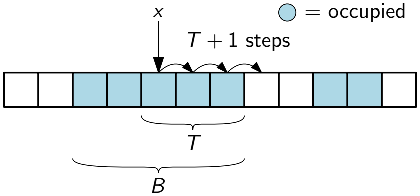


$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$\mathbb{E}[T] \leq \mathbb{E}[B]$

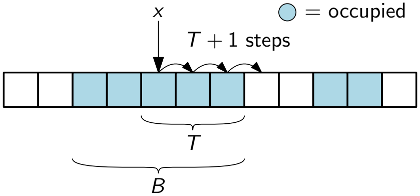


$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$$\mathbb{E}[T] \le \mathbb{E}[B] = \sum_{k \ge 1} k \cdot \Pr[B = k]$$

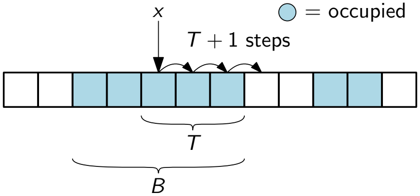


$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$$\mathbb{E}[T] \leq \mathbb{E}[B] = \sum_{k\geq 1} k \cdot \Pr[B = k] = \sum_{k\geq 1} k \cdot \Pr\left[\bigcup_{d=0}^{k-1} A_{h(x)-d,h(x)-d+k-1}\right]$$



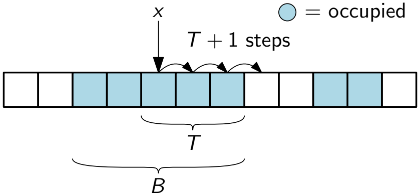
$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$$\mathbb{E}[T] \leq \mathbb{E}[B] = \sum_{k\geq 1} k \cdot \Pr[B = k] = \sum_{k \geq 1} k \cdot \Pr\left[\bigcup_{d=0}^{k-1} A_{h(x)-d,h(x)-d+k-1}\right]$$

$$\overset{(1)}{\leq} \sum_{k\geq 1} k \cdot \sum_{d=0}^{k-1} \Pr\left[A_{h(x)-d,h(x)-d+k-1}\right]$$



$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

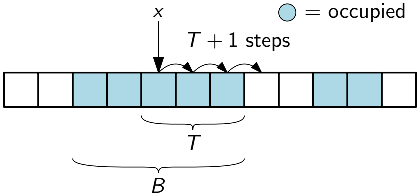
Reasoning:

(1) Union Bound.

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$$\mathbb{E}[T] \le \mathbb{E}[B] = \sum_{k\ge1} k \cdot \Pr[B = k] = \sum_{k\ge1} k \cdot \Pr\left[\bigcup_{d=0}^{k-1} A_{h(x)-d,h(x)-d+k-1}\right]$$

$$\overset{(1)}{\le} \sum_{k\ge1} k \cdot \sum_{d=0}^{k-1} \Pr\left[A_{h(x)-d,h(x)-d+k-1}\right] \overset{(2)}{=} \sum_{k\ge1} k \cdot k \cdot \Pr[A_{1,k}]$$



$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

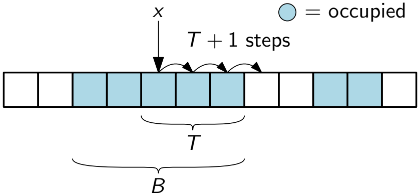
(1) Union Bound.
(2) $h(x)$ is independent of keys in the table and hash distribution is invariant under cyclic shifts.

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$$\mathbb{E}[T] \le \mathbb{E}[B] = \sum_{k\ge1} k \cdot \Pr[B = k] = \sum_{k\ge1} k \cdot \Pr\left[\bigcup_{d=0}^{k-1} A_{h(x)-d,h(x)-d+k-1}\right]$$

$$\overset{(1)}{\le} \sum_{k\ge1} k \cdot \sum_{d=0}^{k-1} \Pr\left[A_{h(x)-d,h(x)-d+k-1}\right] \overset{(2)}{=} \sum_{k\ge1} k \cdot k \cdot \Pr[A_{1,k}]$$

$$\overset{(3)}{\le} \sum_{k\ge1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1,\dots,k\}| \ge k]$$



$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

(1) Union Bound.
(2) $h(x)$ is independent of keys in the table and hash distribution is invariant under cyclic shifts.
(3) Note: Keys stored in block cannot come in from the left.

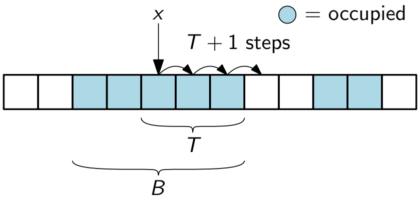
Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$$\mathbb{E}[T] \leq \mathbb{E}[B] = \sum_{k\geq 1} k \cdot \Pr[B = k] = \sum_{k\geq 1} k \cdot \Pr\left[\bigcup_{d=0}^{k-1} A_{h(x)-d,h(x)-d+k-1}\right]$$

$$\overset{(1)}{\leq} \sum_{k\geq 1} k \cdot \sum_{d=0}^{k-1} \Pr\left[A_{h(x)-d,h(x)-d+k-1}\right] \overset{(2)}{=} \sum_{k\geq 1} k \cdot k \cdot \Pr[A_{1,k}]$$

$$\overset{(3)}{\leq} \sum_{k\geq 1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1,\dots,k\}| \geq k]$$

$$\overset{(4)}{\leq} \sum_{k\geq 1} k^2 \cdot \exp(-(1-\alpha)^2 k/3)$$



$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

(1) Union Bound.
(2) $h(x)$ is independent of keys in the table and hash distribution is invariant under cyclic shifts.
(3) Note: Keys stored in block cannot come in from the left.
(4) Chernoff argument from previous slide.

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

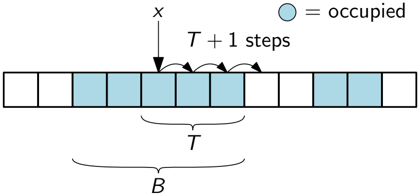
$$\mathbb{E}[T] \le \mathbb{E}[B] = \sum_{k\ge 1} k \cdot \Pr[B = k] = \sum_{k \ge 1} k \cdot \Pr\left[\bigcup_{d=0}^{k-1} A_{h(x)-d,h(x)-d+k-1}\right]$$

$$\overset{(1)}{\le} \sum_{k\ge 1} k \cdot \sum_{d=0}^{k-1} \Pr\left[A_{h(x)-d,h(x)-d+k-1}\right] \overset{(2)}{=} \sum_{k\ge 1} k \cdot k \cdot \Pr[A_{1,k}]$$

$$\overset{(3)}{\le} \sum_{k\ge 1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1,\dots,k\}| \ge k]$$

$$\overset{(4)}{\le} \sum_{k\ge 1} k^2 \cdot \exp(-(1-\alpha)^2 k/3)$$

$$\le \sum_{k\ge 1} k^2 \cdot \exp(-(1-\alpha_{\max})^2 k/3)$$



$A_{u,v}$: $\{u, v\}$ is maximal occupied block:

Reasoning:

(1) Union Bound.
(2) $h(x)$ is independent of keys in the table and hash distribution is invariant under cyclic shifts.
(3) Note: Keys stored in block cannot come in from the left.
(4) Chernoff argument from previous slide.

Proof: Expected LP-Insertion Time under SUHA is $\mathcal{O}(1)$

$$\mathbb{E}[T] \le \mathbb{E}[B] = \sum_{k\ge 1} k \cdot \Pr[B = k] = \sum_{k\ge 1} k \cdot \Pr\left[\bigcup_{d=0}^{k-1} A_{h(x)-d,h(x)-d+k-1}\right]$$

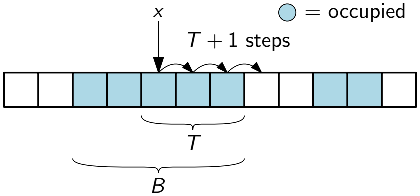
$$\overset{(1)}{\le} \sum_{k\ge 1} k \cdot \sum_{d=0}^{k-1} \Pr\left[A_{h(x)-d,h(x)-d+k-1}\right] \overset{(2)}{=} \sum_{k\ge 1} k \cdot k \cdot \Pr[A_{1,k}]$$

$$\overset{(3)}{\le} \sum_{k\ge 1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1,\dots,k\}| \ge k]$$

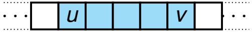
$$\overset{(4)}{\le} \sum_{k\ge 1} k^2 \cdot \exp(-(1-\alpha)^2 k/3)$$

$$\le \sum_{k\ge 1} k^2 \cdot \exp(-(1-\alpha_{\max})^2 k/3) = \mathcal{O}(1).$$

Wolfram Alpha gives: $\int_0^\infty k^2 \exp(-(1-\alpha_{\max})^2 k/3) = \dfrac{54}{(1-\alpha_{\max})^6}$.



$A_{u,v}$: $\{u, v\}$ is maximal occupied block:



Reasoning:

(1) Union Bound.
(2) $h(x)$ is independent of keys in the table and hash distribution is invariant under cyclic shifts.
(3) Note: Keys stored in block cannot come in from the left.
(4) Chernoff argument from previous slide.

Content

Degrees of Independence

(Mutual / Collective) Independence

A family $\mathcal{E}$ of **events** is **independent** if $\forall k \in \mathbb{N}$ and distinct $E_1, \dots, E_k \in \mathcal{E}$ we have

$$\Pr\left[\bigcap_{i=1}^{k} E_i\right] = \prod_{i=1}^{k} \Pr[E_i].$$

A family $\mathcal{X}$ of discrete **random variables** is **independent** if $\forall k \in \mathbb{N}$, distinct $X_1, \dots, X_k \in \mathcal{X}$ and all $x_1, \dots, x_k \in \mathbb{R}$ we have

$$\Pr\left[\bigwedge_{i=1}^{k} X_i = x_i\right] = \prod_{i=1}^{k} \Pr[X_i = x_i].$$

Degrees of Independence

Pairwise Independence

A family $\mathcal{E}$ of **events** is **pairwise independent** if for distinct $E_1, E_2 \in \mathcal{E}$ we have

$$\Pr\left[E_1 \cap E_2\right] = \Pr[E_1] \cdot \Pr[E_2].$$

A family $\mathcal{X}$ of discrete **random variables** is **pairwise independent** if for all distinct $X_1, X_2 \in \mathcal{X}$ and all $x_1, x_2 \in \mathbb{R}$ we have

$$\Pr\left[X_1 = x_1 \wedge X_2 = x_2\right] = \Pr[X_1 = x_1] \cdot \Pr[X_2 = x_2].$$

Degrees of Independence

Karlsruhe Institute of Technology

d-wise Independence

A family $\mathcal{E}$ of **events** is d**-wise independent** if $\forall k \in \{2, \ldots, d\}$ and distinct $E_1, \ldots, E_k \in \mathcal{E}$ we have

$$\Pr\left[\bigcap_{i=1}^{k} E_i\right] = \prod_{i=1}^{k} \Pr[E_i].$$

A family $\mathcal{X}$ of discrete **random variables** is d**-wise independent** if $\forall k \in \{2, \ldots, d\}$, distinct $X_1, \ldots, X_k \in \mathcal{X}$ and all $x_1, \ldots, x_k \in \mathbb{R}$ we have

$$\Pr\left[\bigwedge_{i=1}^{k} X_i = x_i\right] = \prod_{i=1}^{k} \Pr[X_i = x_i].$$

d-Independent Hash Family

Definition: d-Independent Hash Family

A family $\mathcal{H} \subseteq [R]^D$ of hash functions is *d-independent* if for distinct $x_1, \dots, x_d \in D$ and any $i_1, \dots, i_d \in R$: (grey is implied by black)

$$\Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_1) = i_1 \wedge \dots \wedge h(x_d) = i_d] = \prod_{j=1}^{d} \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_j) = i_j] = |R|^{-d}.$$

d-Independent Hash Family

Definition: d-Independent Hash Family

A family $\mathcal{H} \subseteq [R]^D$ of hash functions is *d-independent* if for distinct $x_1, \dots, x_d \in D$ and any $i_1, \dots, i_d \in R$: (grey is implied by black)

$$\Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_1) = i_1 \wedge \ldots \wedge h(x_d) = i_d] = \prod_{j=1}^{d} \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_j) = i_j] = |R|^{-d}.$$

Alternative Definition

$\mathcal{H}$ is d-independent if for $h \sim \mathcal{U}(\mathcal{H})$

- the family $(h(x))_{x \in D}$ of random variables is d-independent *and*
- $h(x) \sim \mathcal{U}(R)$ for each $x \in D$.

d-Independent Hash Family

Karlsruhe Institute of Technology

Definition: d-Independent Hash Family

A family $\mathcal{H} \subseteq [R]^D$ of hash functions is *d-independent* if for distinct $x_1, \ldots, x_d \in D$ and any $i_1, \ldots, i_d \in R$: (grey is implied by black)

$$\Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_1) = i_1 \wedge \ldots \wedge h(x_d) = i_d] = \prod_{j=1}^{d} \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_j) = i_j] = |R|^{-d}.$$

Theorem

Let $D = R = \mathbb{F}$ be a finite field. Then

$$\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \ldots, a_{d-1} \in \mathbb{F}\}$$

is a d-independent family.

Note: $\mathcal{H} \subseteq \mathbb{F}^{\mathbb{F}} \rightsquigarrow$ not yet useful.

Alternative Definition

$\mathcal{H}$ is d-independent if for $h \sim \mathcal{U}(\mathcal{H})$

- the family $(h(x))_{x \in D}$ of random variables is d-independent *and*
- $h(x) \sim \mathcal{U}(R)$ for each $x \in D$.

d-Independent Hash Family

Definition: d-Independent Hash Family

A family $\mathcal{H} \subseteq [R]^D$ of hash functions is *d-independent* if for distinct $x_1, \dots, x_d \in D$ and any $i_1, \dots, i_d \in R$: (grey is implied by black)

$$\Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_1) = i_1 \wedge \dots \wedge h(x_d) = i_d] = \prod_{j=1}^{d} \Pr_{h \sim \mathcal{U}(\mathcal{H})}[h(x_j) = i_j] = |R|^{-d}.$$

Alternative Definition

$\mathcal{H}$ is d-independent if for $h \sim \mathcal{U}(\mathcal{H})$

- the family $(h(x))_{x \in D}$ of random variables is d-independent *and*
- $h(x) \sim \mathcal{U}(R)$ for each $x \in D$.

Theorem

Let $D = R = \mathbb{F}$ be a finite field. Then

$$\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \dots, a_{d-1} \in \mathbb{F}\}$$

is a d-independent family.

Note: $\mathcal{H} \subseteq \mathbb{F}^{\mathbb{F}} \rightsquigarrow$ not yet useful.

Corollary: Smaller Ranges (proof omitted)

- If m divides $|\mathbb{F}|$, then adding "$\bmod m$" gives a d-independent family $\mathcal{H}' \subseteq [m]^{\mathbb{F}}$.
- If m does not divide $|\mathbb{F}|$, then adding "$\bmod m$" gives a family $\mathcal{H}' \subseteq [m]^{\mathbb{F}}$ such that for $h \sim \mathcal{U}(\mathcal{H}')$ the family $(h(x))_{x \in \mathbb{F}}$ is d-independent but only *approximately* uniformly distributed in $[m]$.

Proof: $\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \ldots, a_{d-1} \in \mathbb{F}\}$ is d-independent

Proof: $\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \dots, a_{d-1} \in \mathbb{F}\}$ is d-independent

Let $x_1, \dots, x_d \in \mathbb{F}$ be distinct keys and $i_1, \dots i_d \in \mathbb{F}$ arbitrary.

$\hookrightarrow$ to show : $\Pr_{h \sim \mathcal{U}(\mathcal{H})}[\forall j \in [d] : h(x_j) = i_j] = |\mathbb{F}|^{-d}$.

Proof: $\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \ldots, a_{d-1} \in \mathbb{F}\}$ is d-independent

Let $x_1, \ldots, x_d \in \mathbb{F}$ be distinct keys and $i_1, \ldots i_d \in \mathbb{F}$ arbitrary.

$\hookrightarrow$ to show : $\Pr_{h \sim \mathcal{U}(\mathcal{H})}[\forall j \in [d] : h(x_j) = i_j] = |\mathbb{F}|^{-d}$.

For $h \in \mathcal{H}$ (given via $a_0, \ldots, a_{d-1}$) the following is equivalent:

$$\begin{array}{c} h(x_1) = i_1 \\ h(x_2) = i_2 \\ \vdots \\ h(x_d) = i_d \end{array} \iff \begin{array}{c} a_0 + a_1 x_1 + \cdots + a_{d-1} x_1^{d-1} = i_1 \\ a_0 + a_1 x_2 + \cdots + a_{d-1} x_2^{d-1} = i_2 \\ \vdots \\ a_0 + a_1 x_d + \cdots + a_{d-1} x_d^{d-1} = i_d \end{array}$$

Proof: $\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \ldots, a_{d-1} \in \mathbb{F}\}$ is d-independent

Let $x_1, \ldots, x_d \in \mathbb{F}$ be distinct keys and $i_1, \ldots i_d \in \mathbb{F}$ arbitrary.

$\hookrightarrow$ to show : $\Pr_{h \sim \mathcal{U}(\mathcal{H})}[\forall j \in [d] : h(x_j) = i_j] = |\mathbb{F}|^{-d}$.

For $h \in \mathcal{H}$ (given via $a_0, \ldots, a_{d-1}$) the following is equivalent:

$$\begin{matrix} h(x_1) = i_1 \\ h(x_2) = i_2 \\ \vdots \\ h(x_d) = i_d \end{matrix} \iff \begin{matrix} a_0 + a_1 x_1 + \cdots + a_{d-1} x_1^{d-1} = i_1 \\ a_0 + a_1 x_2 + \cdots + a_{d-1} x_2^{d-1} = i_2 \\ \vdots \\ a_0 + a_1 x_d + \cdots + a_{d-1} x_d^{d-1} = i_d \end{matrix} \iff \underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{d-1} \\ \vdots & & & \ddots & \vdots \\ 1 & x_d & x_d^2 & \ldots & x_d^{d-1} \end{pmatrix}} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_d \end{pmatrix}$$

Proof: $\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \dots, a_{d-1} \in \mathbb{F}\}$ is d-independent

Let $x_1, \dots, x_d \in \mathbb{F}$ be distinct keys and $i_1, \dots i_d \in \mathbb{F}$ arbitrary.

$\hookrightarrow$ to show : $\Pr_{h\sim\mathcal{U}(\mathcal{H})}[\forall j \in [d] : h(x_j) = i_j] = |\mathbb{F}|^{-d}$.

For $h \in \mathcal{H}$ (given via $a_0, \dots, a_{d-1}$) the following is equivalent:

$$\begin{matrix} h(x_1) = i_1 \\ h(x_2) = i_2 \\ \vdots \\ h(x_d) = i_d \end{matrix} \iff \begin{matrix} a_0 + a_1x_1 + \dots + a_{d-1}x_1^{d-1} = i_1 \\ a_0 + a_1x_2 + \dots + a_{d-1}x_2^{d-1} = i_2 \\ \vdots \\ a_0 + a_1x_d + \dots + a_{d-1}x_d^{d-1} = i_d \end{matrix} \iff \underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{d-1} \\ \vdots & & & \ddots & \vdots \\ 1 & x_d & x_d^2 & \dots & x_d^{d-1} \end{pmatrix}}_{\text{Vandermonde matrix } M \Rightarrow \text{ regular}} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_d \end{pmatrix}$$

Proof: $\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \dots, a_{d-1} \in \mathbb{F}\}$ is d-independent

Let $x_1, \dots, x_d \in \mathbb{F}$ be distinct keys and $i_1, \dots i_d \in \mathbb{F}$ arbitrary.

$\hookrightarrow$ to show : $\Pr_{h \sim \mathcal{U}(\mathcal{H})}[\forall j \in [d] : h(x_j) = i_j] = |\mathbb{F}|^{-d}$.

For $h \in \mathcal{H}$ (given via $a_0, \dots, a_{d-1}$) the following is equivalent:

$$\begin{matrix} h(x_1) = i_1 \\ h(x_2) = i_2 \\ \vdots \\ h(x_d) = i_d \end{matrix} \iff \begin{matrix} a_0 + a_1 x_1 + \dots + a_{d-1} x_1^{d-1} = i_1 \\ a_0 + a_1 x_2 + \dots + a_{d-1} x_2^{d-1} = i_2 \\ \vdots \\ a_0 + a_1 x_d + \dots + a_{d-1} x_d^{d-1} = i_d \end{matrix} \iff \underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{d-1} \\ \vdots & & & \ddots & \vdots \\ 1 & x_d & x_d^2 & \dots & x_d^{d-1} \end{pmatrix}}_{\text{Vandermonde matrix } M \Rightarrow \text{ regular}} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_d \end{pmatrix}$$

Exactly one vector $\vec{a} = M^{-1} \cdot \vec{i}$ solves the equation.

Proof: $\mathcal{H} := \{x \mapsto \sum_{i=0}^{d-1} a_i x^i \mid a_0, \ldots, a_{d-1} \in \mathbb{F}\}$ is d-independent

Let $x_1, \ldots, x_d \in \mathbb{F}$ be distinct keys and $i_1, \ldots i_d \in \mathbb{F}$ arbitrary.

$\hookrightarrow$ to show : $\Pr_{h \sim \mathcal{U}(\mathcal{H})}[\forall j \in [d] : h(x_j) = i_j] = |\mathbb{F}|^{-d}$.

For $h \in \mathcal{H}$ (given via $a_0, \ldots, a_{d-1}$) the following is equivalent:

$$\begin{array}{c} h(x_1) = i_1 \\ h(x_2) = i_2 \\ \vdots \\ h(x_d) = i_d \end{array} \iff \begin{array}{c} a_0 + a_1 x_1 + \cdots + a_{d-1} x_1^{d-1} = i_1 \\ a_0 + a_1 x_2 + \cdots + a_{d-1} x_2^{d-1} = i_2 \\ \vdots \\ a_0 + a_1 x_d + \cdots + a_{d-1} x_d^{d-1} = i_d \end{array} \iff \underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{d-1} \\ \vdots & & & \ddots & \vdots \\ 1 & x_d & x_d^2 & \ldots & x_d^{d-1} \end{pmatrix}}_{\text{Vandermonde matrix } M \Rightarrow \text{ regular}} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_d \end{pmatrix}$$

Exactly one vector $\vec{a} = M^{-1} \cdot \vec{i}$ solves the equation.

$$\Rightarrow \Pr_{h \sim \mathcal{U}(\mathcal{H})}[\forall j : h(x_j) = i_j] = \Pr_{a_0, \ldots, a_{d-1} \sim \mathcal{U}(\mathbb{F})}[\vec{a} = M^{-1} \cdot \vec{i}\,] = \mathbb{F}^{-d}. \quad \square$$

Concentration Bound for d-Independent Variables

(Tricky) Exercise

Let d be even and $X_1, \ldots, X_n \sim Ber(p)$ a d-independent family of random variables with $p = \Omega(1/n)$.
Let $X = \sum_{i=1}^{n} X_i$. Then for any $\varepsilon > 0$ we have

$$\Pr[X - \mathbb{E}[X] \geq \varepsilon \mathbb{E}[X]] = \mathcal{O}(\varepsilon^{-d} \mathbb{E}[X]^{-d/2}).$$

Concentration Bound for d-Independent Variables

(Tricky) Exercise

Let d be even and $X_1, \dots, X_n \sim Ber(p)$ a d-independent family of random variables with $p = \Omega(1/n)$.
Let $X = \sum_{i=1}^{n} X_i$. Then for any $\varepsilon > 0$ we have

$$\Pr[X - \mathbb{E}[X] \geq \varepsilon \mathbb{E}[X]] = \mathcal{O}(\varepsilon^{-d} \mathbb{E}[X]^{-d/2}).$$

Remark: Weaker than Chernoff, stronger than Chebyshev

Chebycheff gives $\Pr[X - \mathbb{E}[X] \geq \varepsilon \mathbb{E}[X]] \leq \frac{1-p}{\varepsilon^2 \mathbb{E}[X]}$. (requires $d = 2$)
Chernoff gave $\Pr[X - \mathbb{E}[X] \geq \varepsilon \mathbb{E}[X]] \leq \exp(-\varepsilon^2 \mathbb{E}[X]/3)$. (requires $d = n$).

Preparation: A Concentration Bound

again for *d*-independence

Lemma (last slide)

For d-independent $X_1, \ldots, X_n \sim Ber(p)$ and $X = \sum_{i \in [n]} X_i$ we have $\Pr[X \geq (1+\varepsilon)\mathbb{E}[X]] = \mathcal{O}(\varepsilon^{-d}\mathbb{E}[X]^{-d/2})$.

Preparation: A Concentration Bound

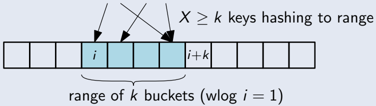
again for *d*-independence

Karlsruhe Institute of Technology

Lemma (last slide)

For d-independent $X_1, \ldots, X_n \sim Ber(p)$ and $X = \sum_{i \in [n]} X_i$ we have $\Pr[X \geq (1+\varepsilon)\mathbb{E}[X]] = \mathcal{O}(\varepsilon^{-d}\mathbb{E}[X]^{-d/2})$.

Lemma: $\geq k$ hits in segment of length k



Let $\mathcal{H}$ be a d-independent hash family and $h \sim \mathcal{U}(\mathcal{H})$.
Let $k \in \mathbb{N}$ and $X = |\{y \in S \mid h(y) \in \{1, \ldots, k\}\}|$.

Then $\Pr[X \geq k] \leq \mathcal{O}((1-\alpha)^{-d} k^{-d/2})$.

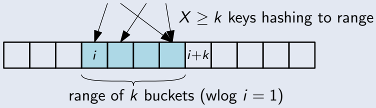
Preparation: A Concentration Bound

again for *d*-independence

Lemma (last slide)

For d-independent $X_1, \ldots, X_n \sim Ber(p)$ and $X = \sum_{i \in [n]} X_i$ we have $\Pr[X \geq (1+\varepsilon)\mathbb{E}[X]] = \mathcal{O}(\varepsilon^{-d}\mathbb{E}[X]^{-d/2})$.

Lemma: $\geq k$ hits in segment of length k



Let $\mathcal{H}$ be a d-independent hash family and $h \sim \mathcal{U}(\mathcal{H})$.
Let $k \in \mathbb{N}$ and $X = |\{y \in S \mid h(y) \in \{1, \ldots, k\}\}|$.

Then $\Pr[X \geq k] \leq \mathcal{O}((1-\alpha)^{-d} k^{-d/2})$.

Proof

Let $S = \{x_1, \ldots, x_n\}$ and $X_i = \mathbb{1}_{\{h(x_i) \in \{1,\ldots,k\}\}} \sim Ber(\frac{k}{m})$.
Then $X = \sum_{i \in [n]} X_i$ fits the Lemma with $\mathbb{E}[X] = \frac{kn}{m} = \alpha k$.

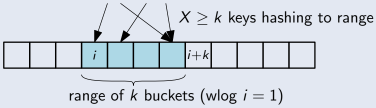
Preparation: A Concentration Bound

again for d-independence

Lemma (last slide)

For d-independent $X_1, \dots, X_n \sim Ber(p)$ and $X = \sum_{i \in [n]} X_i$ we have $\Pr[X \geq (1+\varepsilon)\mathbb{E}[X]] = \mathcal{O}(\varepsilon^{-d}\mathbb{E}[X]^{-d/2})$.

Lemma: $\geq k$ hits in segment of length k



Let $\mathcal{H}$ be a d-independent hash family and $h \sim \mathcal{U}(\mathcal{H})$.
Let $k \in \mathbb{N}$ and $X = |\{y \in S \mid h(y) \in \{1, \dots, k\}\}|$.

Then $\Pr[X \geq k] \leq \mathcal{O}((1-\alpha)^{-d}k^{-d/2})$.

Proof

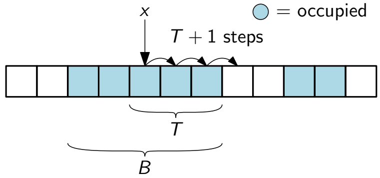
Let $S = \{x_1, \dots, x_n\}$ and $X_i = \mathbb{1}_{\{h(x_i) \in \{1,\dots,k\}\}} \sim Ber(\frac{k}{m})$.
Then $X = \sum_{i \in [n]} X_i$ fits the Lemma with $\mathbb{E}[X] = \frac{kn}{m} = \alpha k$.

$$\begin{aligned}
\Pr[X \geq k] &= \Pr[X \geq \tfrac{1}{\alpha}\mathbb{E}[X]] \\
&= \Pr[X \geq (1 + \tfrac{1-\alpha}{\alpha})\mathbb{E}[X]] \\
&= \mathcal{O}\left(\left(\frac{1-\alpha}{\alpha}\right)^{-d}(\alpha k)^{-d/2}\right) \\
&\leq \mathcal{O}((1-\alpha)^{-d}k^{-d/2}). \quad \text{(using } \alpha \leq 1\text{)}
\end{aligned}$$

Theorem: Linear Probing with d-independence

Under the same conditions as before, except with 9-independent hash functions, the insertion time $T_{n,m}$ for linear probing satisfies:

$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1)$$



$A_{u,v}$: $\{u, v\}$ is a maximal occupied block:

Reasoning:

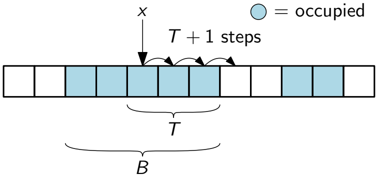
Theorem: Linear Probing with d-independence

Under the same conditions as before, except with 9-independent hash functions, the insertion time $T_{n,m}$ for linear probing satisfies:

$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1)$$

Proof Sketch

$\mathbb{E}[T]$



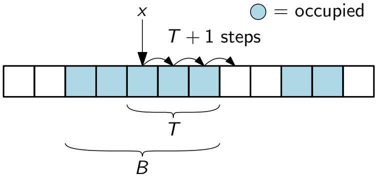
$A_{u,v}$: $\{u, v\}$ is a maximal occupied block:

Reasoning:

Theorem: Linear Probing with d-independence

Under the same conditions as before, except with 9-independent hash functions, the insertion time $T_{n,m}$ for linear probing satisfies:

$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1)$$



$A_{u,v}$: $\{u, v\}$ is a maximal occupied block:

Reasoning:

Proof Sketch

$$\mathbb{E}[T] \leq \mathbb{E}[B]$$

Theorem: Linear Probing with d-independence

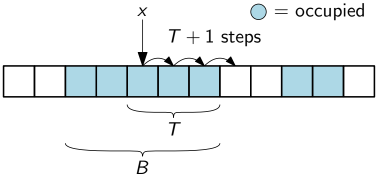
Under the same conditions as before, except with 9-independent hash functions, the insertion time $T_{n,m}$ for linear probing satisfies:

$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1)$$

Proof Sketch

$$\mathbb{E}[T] \leq \mathbb{E}[B] \leq \dots$$

$$\overset{(1)}{\leq} \sum_{k \geq 1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1, \dots, k\}\}| \geq k]$$



$A_{u,v}$: $\{u, v\}$ is a maximal occupied block:

Reasoning:

(1) Same as before, except we have to condition on $h(x)$ and may only use 8-independence in the following. (this is the hand wavy part!)

Theorem: Linear Probing with d-independence

Under the same conditions as before, except with 9-independent hash functions, the insertion time $T_{n,m}$ for linear probing satisfies:

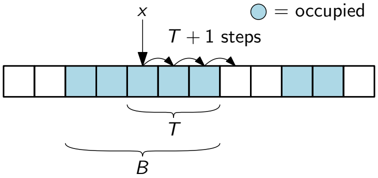
$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1)$$

Proof Sketch

$$\mathbb{E}[T] \leq \mathbb{E}[B] \leq \ldots$$

$$\stackrel{(1)}{\leq} \sum_{k \geq 1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1, \ldots, k\}\}| \geq k]$$

$$\stackrel{(2)}{\leq} \sum_{k \geq 1} k^2 \cdot \mathcal{O}((1-\alpha)^{-8} k^{-8/2})$$



$A_{u,v}$: $\{u, v\}$ is a maximal occupied block:

Reasoning:

(1) Same as before, except we have to condition on $h(x)$ and may only use 8-independence in the following. (this is the hand wavy part!)

(2) Concentration bound from previous slide for $d = 8$.

Theorem: Linear Probing with d-independence

Under the same conditions as before, except with 9-independent hash functions, the insertion time $T_{n,m}$ for linear probing satisfies:

$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1)$$

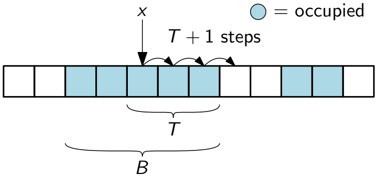
Proof Sketch

$$\mathbb{E}[T] \leq \mathbb{E}[B] \leq \dots$$

$$\stackrel{(1)}{\leq} \sum_{k \geq 1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1, \dots, k\}\}| \geq k]$$

$$\stackrel{(2)}{\leq} \sum_{k \geq 1} k^2 \cdot \mathcal{O}((1-\alpha)^{-8} k^{-8/2})$$

$$\leq \sum_{k \geq 1} k^{-2} \cdot \mathcal{O}((1-\alpha)^{-8})$$



$A_{u,v}$: $\{u, v\}$ is a maximal occupied block:

Reasoning:

(1) Same as before, except we have to condition on $h(x)$ and may only use 8-independence in the following. (this is the hand wavy part!)

(2) Concentration bound from previous slide for $d = 8$.

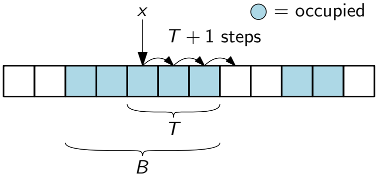
Theorem: Linear Probing with d-independence

Under the same conditions as before, except with 9-independent hash functions, the insertion time $T_{n,m}$ for linear probing satisfies:

$$\mathbb{E}[T_{n,m}] = \mathcal{O}(1)$$

Proof Sketch

$$\begin{aligned}
\mathbb{E}[T] \le \mathbb{E}[B] &\le \dots \\
&\overset{(1)}{\le} \sum_{k\ge 1} k^2 \cdot \Pr[|\{y \in S \mid h(y) \in \{1,\dots,k\}\}| \ge k] \\
&\overset{(2)}{\le} \sum_{k\ge 1} k^2 \cdot \mathcal{O}((1-\alpha)^{-8} k^{-8/2}) \\
&\le \sum_{k\ge 1} k^{-2} \cdot \mathcal{O}((1-\alpha)^{-8}) \\
&\overset{(3)}{=} \tfrac{\pi^2}{6}\mathcal{O}((1-\alpha)^{-8}) = \mathcal{O}(1). \quad \square
\end{aligned}$$



$A_{u,v}$: $\{u, v\}$ is a maximal occupied block:

Reasoning:

(1) Same as before, except we have to condition on $h(x)$ and may only use 8-independence in the following. (this is the hand wavy part!)

(2) Concentration bound from previous slide for $d = 8$.

(3) If interested, see 3Blue1Brown video: `https://www.youtube.com/watch?v=d-o3eB9sfls`

Final Remarks on Linear Probing + Universal Hashing

Karlsruhe Institute of Technology

Much more is known about insertion times of linear probing:

- Any 5-independent family gives $\mathcal{O}(\frac{1}{(1-\alpha)^2})$.
  $\hookrightarrow$ A. Pagh, R. Pagh, and Ruzic 2011
- An (artificially bad) 4-independent family gives $\Omega(\log n)$.
  $\hookrightarrow$ Pătraşcu and Thorup 2016
- A (well-designed) 4-independent family gives $\mathcal{O}(\frac{1}{(1-\alpha)^2})$.
  $\hookrightarrow$ Puatracscu and Thorup 2013

Conclusion

Technical Takeaway: Performance of Hash Tables

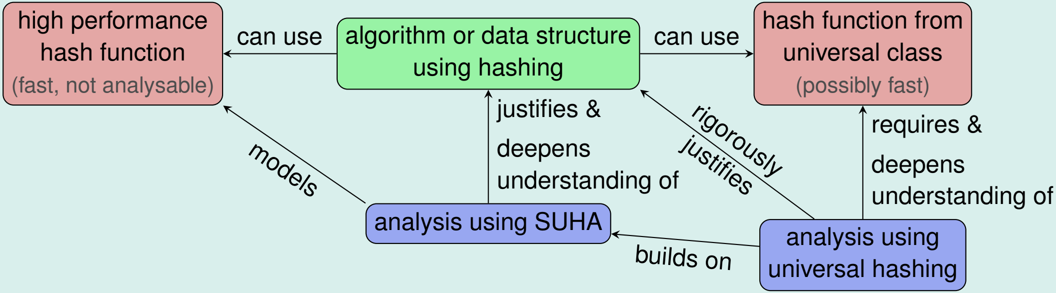
For both an **ideal hash function** (SUHA) and a random hash function from a suitable **universal class**, a hash table using **linear probing** or **chaining** provably has an expected running time of $\mathcal{O}(1)$ per operation.

Conclusion

Technical Takeaway: Performance of Hash Tables

For both an **ideal hash function** (SUHA) and a random hash function from a suitable **universal class**, a hash table using **linear probing** or **chaining** provably has an expected running time of $\mathcal{O}(1)$ per operation.

Non-Technical Takeaway: Approaches to analyse hashing based algorithms

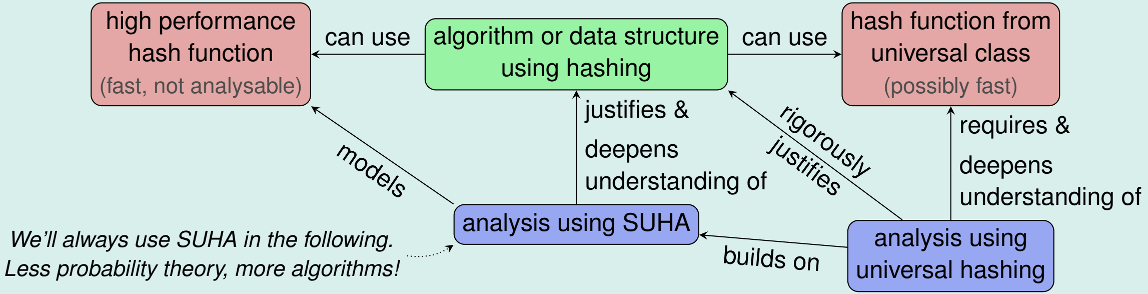


Conclusion

Technical Takeaway: Performance of Hash Tables

For both an **ideal hash function** (SUHA) and a random hash function from a suitable **universal class**, a hash table using **linear probing** or **chaining** provably has an expected running time of $\mathcal{O}(1)$ per operation.

Non-Technical Takeaway: Approaches to analyse hashing based algorithms



Anhang: Mögliche Prüfungsfragen I

Karlsruhe Institute of Technology

- Was könnte eine Idealvorstellung einer Hashfunktion sein? Inwiefern wäre eine ideale Hashfunktion nützlich? Was ist das Problem an dieser Vorstellung?
- Was ist die Simple Uniform Hashing Assumption (SUHA)? Was spricht dafür diese Annahme zu treffen? Welche Alternativen gibt es?
- Inwiefern ist eine pseudozufällige Funktion mit kryptographischen Ununterscheidbarkeitsgarantien nützlich für uns? Wie ist der Zusammenhang zur SUHA?*
- Universelles Hashing:
  - Wie ist c-Universalität definiert?
  - Welche c-universellen Hashklasse haben wir kennengelernt? Wie haben wir die c-Universalität bewiesen?
  - Wie ist d-Unabhängigkeit für eine Hashklasse definiert?
  - Welche d-universelle Hashklasse haben wir kennengelernt?
  - Welcher Zusammenhang besteht zwischen d-Unabhängigkeit und c-Universalität? (Übungsaufgabe)
  - Chernoff Schranken sind für Summen unabhängiger Zufallsvariablen gedacht. Was kann man machen, wenn die Zufallsvariablen nur d-unabhängig sind?*

Anhang: Mögliche Prüfungsfragen II

- Betrachten wir Hashing mit verketteten Listen:
  - Welche Schranke an die erwartete Einfügezeit haben wir bewiesen? Wie?
  - An welcher Stelle spielt die Verteilung der Hashfunktion eine Rolle?
  - Nenne eine hinreichende Eigenschaft, die eine universelle Hashklasse haben sollte, damit der Beweis funktioniert.
- Betrachten wir Hashing mit linearem Sondieren:
  - Welche Schranke an die erwartete Laufzeit haben wir bewiesen? Wie?
  - An welcher Stelle spielt die Verteilung der Hashfunktion eine Rolle?
  - Nenne eine hinreichende Eigenschaft, die eine universelle Hashklasse haben sollte, damit der Beweis funktioniert.
  - Wie wir diese Eigenschaft ausgenutzt?*

References I

[1] Anna Pagh, Rasmus Pagh, and Milan Ruzic. "Linear Probing with 5-wise Independence". In: *SIAM Rev.* 53.3 (2011), pp. 547–558. DOI: 10.1137/110827831. URL: https://doi.org/10.1137/110827831.

[2] Mihai Pătraşcu and Mikkel Thorup. "On the *k*-Independence Required by Linear Probing and Minwise Independence". In: *ACM Trans. Algorithms* 12.1 (2016), 8:1–8:27. DOI: 10.1145/2716317. URL: https://doi.org/10.1145/2716317.

[3] Mihai Puatracscu and Mikkel Thorup. "Twisted Tabulation Hashing". In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. Ed. by Sanjeev Khanna. SIAM, 2013, pp. 209–228. DOI: 10.1137/1.9781611973105.16. URL: https://doi.org/10.1137/1.9781611973105.16.

[4] Mikkel Thorup. "High Speed Hashing for Integers and Strings". In: *CoRR* abs/1504.06804 (2015). arXiv: 1504.06804. URL: http://arxiv.org/abs/1504.06804.