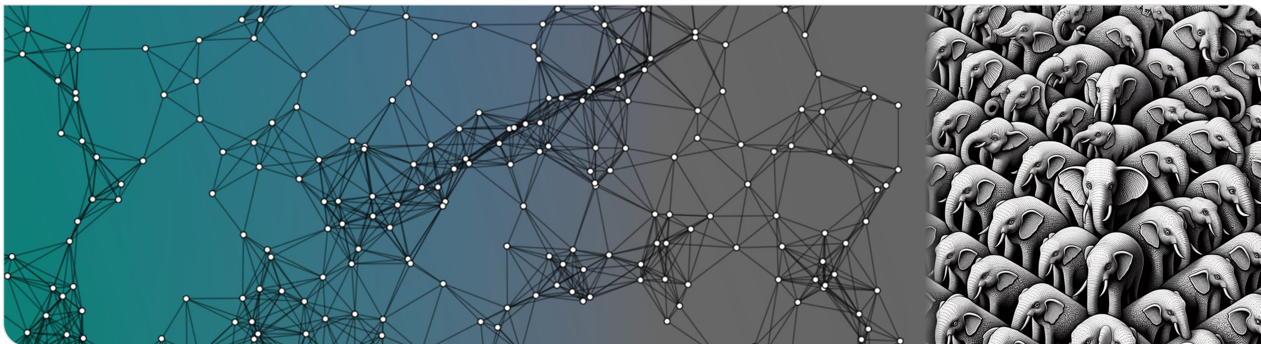
Karlsruhe Institute of Technology

Algorithm Engineering

Probability and Computing – Retrieval and Perfect Hashing

Stefan Walzer, Maximilian Katzmann | WS 2023/2024



Content

Karlsruhe Institute of Technology

1. Retrieval Data Structures

- The Retrieval Problem
- Motivation
- Construction Using Peeling

2. (Minimal-) Perfect Hashing

- The Perfect Hashing Problem
- Motivation: Updatable Retrieval
- Construction using Trial and Error
- Construction using Cuckoo Hashing and Retrieval

Notational heads-up

Karlsruhe Institute of Technology

- in other chapters $[k] := \{1, \ldots, k\}$
- in this chapter *sometimes* $[k] := \{0, \ldots, k-1\}$
- you'll figure it out...

The Retrieval Problem

Karlsruhe Institute of Technology

The retrieval data type (for universe D, range $[k]$)

construct(f):

input: function $f : S \to [k]$ //$f \subseteq D \times [k]$
where $S \subseteq D$ has size $n = |S|$

output: data structure R.

eval(R, x):

input: $R =$ **construct**($f : S \to [k]$), $x \in D$

output: some value in $[k]$

requirement: **eval**(R, x) $= f(x)$ for all $x \in S$

The Retrieval Problem

Karlsruhe Institute of Technology

The retrieval data type (for universe D, range $[k]$)

construct(f):

- input: function $f : S \to [k]$ //$f \subseteq D \times [k]$
  where $S \subseteq D$ has size $n = |S|$
- output: data structure R.

eval(R, x):

- input: $R =$ **construct**($f : S \to [k]$), $x \in D$
- output: some value in $[k]$
- requirement: **eval**(R, x) $= f(x)$ for all $x \in S$

Goals

- space requirement of R is $\mathcal{O}(n \log k)$ bits
  - possibly even $n\lceil \log_2(k) \rceil + o(n)$
  - ⚠ naively storing f needs $\Omega(n(\log(k) + \log(|D|)))$
- ideally running time of **eval** is $\mathcal{O}(1)$
- ideally running time of **construct** is $\mathcal{O}(n)$

The Retrieval Problem

Karlsruhe Institute of Technology

The retrieval data type (for universe D, range $[k]$)

construct(f):

- input: function $f : S \to [k]$ //$f \subseteq D \times [k]$ where $S \subseteq D$ has size $n = |S|$
- output: data structure R.

eval(R, x):

- input: $R = \textbf{construct}(f : S \to [k])$, $x \in D$
- output: some value in $[k]$
- requirement: $\textbf{eval}(R, x) = f(x)$ for all $x \in S$

The price to pay

- R cannot be used to decide "is $x \in S$?"
- **eval**(R, x) is *unspecified* if $x \notin S$.

Goals

- space requirement of R is $\mathcal{O}(n \log k)$ bits
  - possibly even $n\lceil \log_2(k) \rceil + o(n)$
  - ⚠ naively storing f needs $\Omega(n(\log(k) + \log(|D|)))$
- ideally running time of **eval** is $\mathcal{O}(1)$
- ideally running time of **construct** is $\mathcal{O}(n)$

The Retrieval Problem

Karlsruhe Institute of Technology

The retrieval data type (for universe D, range $[k]$)

construct(f):

input: function $f : S \to [k]$ //$f \subseteq D \times [k]$
where $S \subseteq D$ has size $n = |S|$

output: data structure R.

eval(R, x):

input: $R =$ **construct**$(f : S \to [k])$, $x \in D$

output: some value in $[k]$

requirement: **eval**$(R, x) = f(x)$ for all $x \in S$

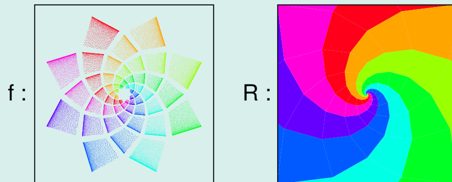
The price to pay

- R cannot be used to decide "is $x \in S$?"
- **eval**(R, x) is *unspecified* if $x \notin S$.

Goals

- space requirement of R is $\mathcal{O}(n \log k)$ bits
  - possibly even $n\lceil\log_2(k)\rceil + o(n)$
  - ⚠ naively storing f needs $\Omega(n(\log(k) + \log(|D|)))$
- ideally running time of **eval** is $\mathcal{O}(1)$
- ideally running time of **construct** is $\mathcal{O}(n)$

Intuition



- R is a *continuation* of f
- information about the domain S is lost.

Motivation for Retrieval

Karlsruhe Institute of Technology

Task: Predict gender based on first name

First name:
John
Last name:
Doe

Gender:

○ F ○ M ○ other

- want $\geq 90\%$ accuracy
- client side only
- lightweight

Motivation for Retrieval

Karlsruhe Institute of Technology

Task: Predict gender based on first name

First name:
John
Last name:
Doe
Gender:
F M other

- want $\geq 90\%$ accuracy
- client side only
- lightweight

Have large data base:

Annotated list of 10000 most common first names.

$$f : \{\text{Dave} \mapsto M, \text{Joanna} \mapsto F, \text{Christina} \mapsto F, \ldots\}$$

$\approx$ 10 bytes per name, too large to send to client.

Motivation for Retrieval

Karlsruhe Institute of Technology

Task: Predict gender based on first name

First name: John
Last name: Doe
Gender:
○ F ○ M ○ other

- want $\geq 90\%$ accuracy
- client side only
- lightweight

Have large data base:

Annotated list of 10000 most common first names.

$$f : \{\text{Dave} \mapsto M, \text{Joanna} \mapsto F, \text{Christina} \mapsto F, \ldots\}$$

$\approx$ 10 bytes per name, too large to send to client.

Solution using retrieval

- send $R = \textbf{construct}(f)$ to client
  $\hookrightarrow \approx 1$ bit per name
- prefill gender with $\textbf{eval}(R, \text{firstName})$

Motivation for Retrieval

Karlsruhe Institute of Technology

Task: Predict gender based on first name

First name: John
Last name: Doe
Gender:
○ F ○ M ○ other

- want $\geq 90\%$ accuracy
- client side only
- lightweight

Have large data base:

Annotated list of 10000 most common first names.

$$f : \{\text{Dave} \mapsto M, \text{Joanna} \mapsto F, \text{Christina} \mapsto F, \dots\}$$

$\approx$ 10 bytes per name, too large to send to client.

Solution using retrieval

- send $R = \textbf{construct}(f)$ to client
  $\hookrightarrow \approx 1$ bit per name
- prefill gender with $\textbf{eval}(R, \text{firstName})$

Weaknesses:

May guess incorrectly if

- name is ambiguous (“Kim”, “Chris”)
- user is non-binary / prefers not to say
- name not listed in f (e.g. “Crhistina”, “Inghean”)
  $\hookrightarrow$ would be better to *refrain from guessing*

Content

Karlsruhe Institute of Technology

1. Retrieval Data Structures

- The Retrieval Problem
- Motivation
- Construction Using Peeling

2. (Minimal-) Perfect Hashing

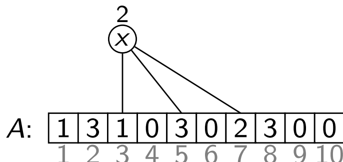
- The Perfect Hashing Problem
- Motivation: Updatable Retrieval
- Construction using Trial and Error
- Construction using Cuckoo Hashing and Retrieval

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\textbf{eval}(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$



Peeling $\rightarrow$ Cuckoo-Style Retrieval

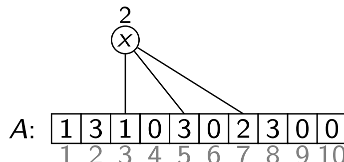
Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$



Peeling $\to$ Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

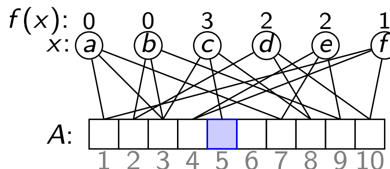
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

$\hookrightarrow$ can forget about x_i "for now" and focus on the rest

$\hookrightarrow$ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\textbf{eval}(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

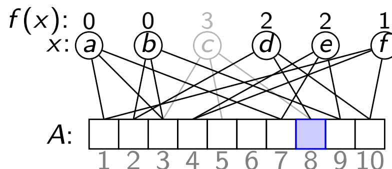
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i “for now” and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$

d: $A[8] := 2 - A[2] - A[10]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

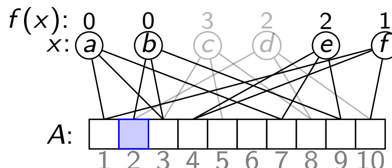
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i “for now” and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$
d: $A[8] := 2 - A[2] - A[10]$
b: $A[2] := 0 - A[3] - A[9]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

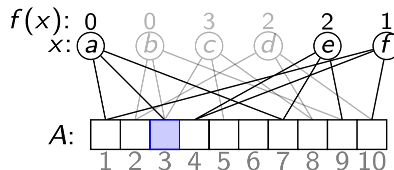
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i "for now" and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$
d: $A[8] := 2 - A[2] - A[10]$
b: $A[2] := 0 - A[3] - A[9]$
a: $A[3] := 0 - A[1] - A[7]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

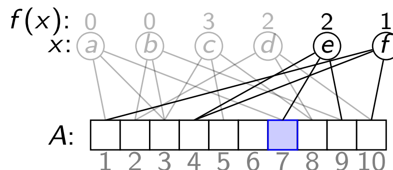
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i “for now” and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$
d: $A[8] := 2 - A[2] - A[10]$
b: $A[2] := 0 - A[3] - A[9]$
a: $A[3] := 0 - A[1] - A[7]$
e: $A[7] := 2 - A[4] - A[9]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

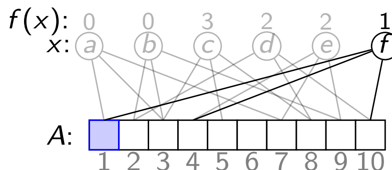
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i "for now" and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$
d: $A[8] := 2 - A[2] - A[10]$
b: $A[2] := 0 - A[3] - A[9]$
a: $A[3] := 0 - A[1] - A[7]$
e: $A[7] := 2 - A[4] - A[9]$
f: $A[1] := 1 - A[4] - A[10]$

Peeling → Cuckoo-Style Retrieval

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

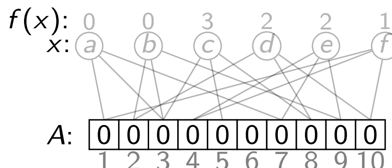
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i "for now" and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$
d: $A[8] := 2 - A[2] - A[10]$
b: $A[2] := 0 - A[3] - A[9]$
a: $A[3] := 0 - A[1] - A[7]$
e: $A[7] := 2 - A[4] - A[9]$
f: $A[1] := 1 - A[4] - A[10]$

Peeling $\to$ Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

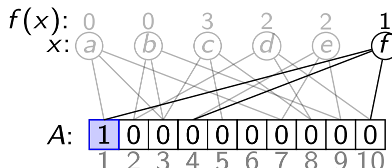
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i “for now” and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$

d: $A[8] := 2 - A[2] - A[10]$

b: $A[2] := 0 - A[3] - A[9]$

a: $A[3] := 0 - A[1] - A[7]$

e: $A[7] := 2 - A[4] - A[9]$

f: $A[1] := 1 - A[4] - A[10]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

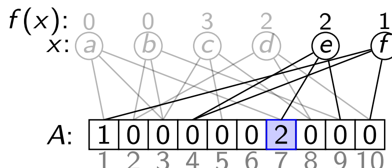
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i "for now" and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$
d: $A[8] := 2 - A[2] - A[10]$
b: $A[2] := 0 - A[3] - A[9]$
a: $A[3] := 0 - A[1] - A[7]$
e: $A[7] := 2 - A[4] - A[9]$
f: $A[1] := 1 - A[4] - A[10]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

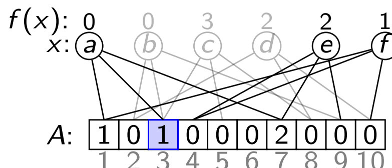
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i "for now" and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$

d: $A[8] := 2 - A[2] - A[10]$

b: $A[2] := 0 - A[3] - A[9]$

a: $A[3] := 0 - A[1] - A[7]$

e: $A[7] := 2 - A[4] - A[9]$

f: $A[1] := 1 - A[4] - A[10]$

Peeling $\to$ Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\textbf{eval}(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

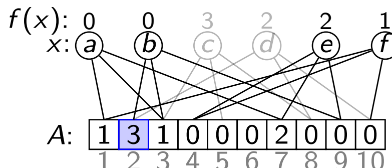
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

$\hookrightarrow$ can forget about x_i “for now” and focus on the rest

$\hookrightarrow$ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$

d: $A[8] := 2 - A[2] - A[10]$

b: $A[2] := 0 - A[3] - A[9]$

a: $A[3] := 0 - A[1] - A[7]$

e: $A[7] := 2 - A[4] - A[9]$

f: $A[1] := 1 - A[4] - A[10]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

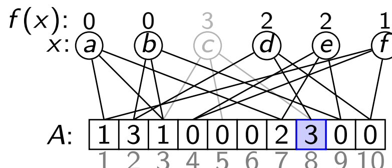
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i “for now” and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$

d: $A[8] := 2 - A[2] - A[10]$

b: $A[2] := 0 - A[3] - A[9]$

a: $A[3] := 0 - A[1] - A[7]$

e: $A[7] := 2 - A[4] - A[9]$

f: $A[1] := 1 - A[4] - A[10]$

Peeling → Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

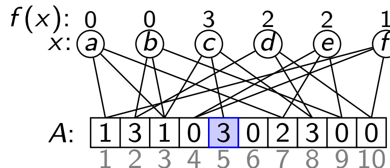
- space $1.23n\lceil\log_2(k)\rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

↪ can forget about x_i "for now" and focus on the rest

↪ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$

d: $A[8] := 2 - A[2] - A[10]$

b: $A[2] := 0 - A[3] - A[9]$

a: $A[3] := 0 - A[1] - A[7]$

e: $A[7] := 2 - A[4] - A[9]$

f: $A[1] := 1 - A[4] - A[10]$

Peeling $\to$ Cuckoo-Style Retrieval

Karlsruhe Institute of Technology

Retrieval Data Structure $R = (h_1, h_2, h_3, A)$

- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

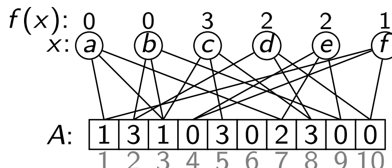
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

How does **construct**(f) choose A?

If $A[j]$ is only used by x_i then setting $A[j]$ *in the end* takes care of x_i without affecting other keys.

$\hookrightarrow$ can forget about x_i "for now" and focus on the rest

$\hookrightarrow$ if configuration is peelable, this takes care of all keys



Equations (mod k for $k = 4$)

c: $A[5] := 3 - A[3] - A[8]$
d: $A[8] := 2 - A[2] - A[10]$
b: $A[2] := 0 - A[3] - A[9]$
a: $A[3] := 0 - A[1] - A[7]$
e: $A[7] := 2 - A[4] - A[9]$
f: $A[1] := 1 - A[4] - A[10]$

Content

1. Retrieval Data Structures

- The Retrieval Problem
- Motivation
- Construction Using Peeling

2. (Minimal-) Perfect Hashing

- The Perfect Hashing Problem
- Motivation: Updatable Retrieval
- Construction using Trial and Error
- Construction using Cuckoo Hashing and Retrieval

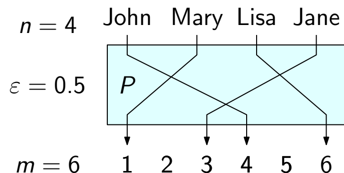
The Perfect Hashing Problem

Karlsruhe Institute of Technology

Perfect hashing data type (for universe D, $\varepsilon \geq 0$)

construct(S):
- input: $S \subseteq D$ of size $n = |S|$
- output: data structure P.

eval(P, x):
- input: $P = \textbf{construct}(S)$ and $x \in D$
- output: a number in $[m]$ where $m = (1 + \varepsilon)n$
- requirement: $x \mapsto \textbf{eval}(P, x)$ is injective on S



The Perfect Hashing Problem

Karlsruhe Institute of Technology

Perfect hashing data type (for universe D, $\varepsilon \geq 0$)

construct(S):

input: $S \subseteq D$ of size $n = |S|$

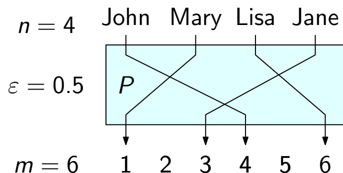
output: data structure P.

eval(P, x):

input: $P = \textbf{construct}(S)$ and $x \in D$

output: a number in $[m]$ where $m = (1 + \varepsilon)n$

requirement: $x \mapsto \textbf{eval}(P, x)$ is injective on S



Goals

- ε is small // $\varepsilon = 0$: *Minimal* perfect hashing
- space requirement of P is $\mathcal{O}(n)$ bits
  - $\approx 1.44n$ bits is necessary and sufficient for $\varepsilon = 0$
  - note: storing S might need $\Omega(n \log(|D|))$ bits.
- ideally: running time of **eval** is $\mathcal{O}(1)$
- ideally: running time of **construct** is $\mathcal{O}(n)$

The Perfect Hashing Problem

Karlsruhe Institute of Technology

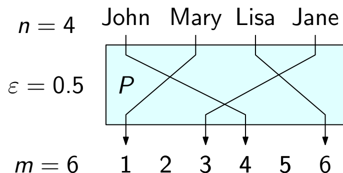
Perfect hashing data type (for universe D, $\varepsilon \geq 0$)

construct(S):
- input: $S \subseteq D$ of size $n = |S|$
- output: data structure P.

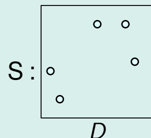
eval(P, x):
- input: $P =$ **construct**(S) and $x \in D$
- output: a number in $[m]$ where $m = (1 + \varepsilon)n$
- requirement: $x \mapsto$ **eval**(P, x) is injective on S

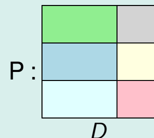
Goals

- ε is small // $\varepsilon = 0$: *Minimal* perfect hashing
- space requirement of P is $\mathcal{O}(n)$ bits
  - $\approx 1.44n$ bits is necessary and sufficient for $\varepsilon = 0$
  - note: storing S might need $\Omega(n \log(|D|))$ bits.
- ideally: running time of **eval** is $\mathcal{O}(1)$
- ideally: running time of **construct** is $\mathcal{O}(n)$



Intuition

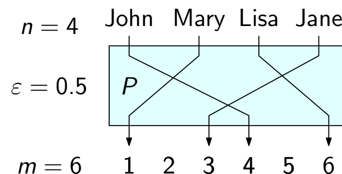
S :


P :


- P is partition of D that separates S
- details about S are lost.
- note: P is “perfect hash function” but need not be random

Motivation for (Minimum-) Perfect Hashing

Karlsruhe Institute of Technology

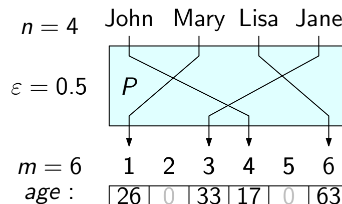


Short IDs

Replace keys with short unique identifies

$$\textbf{eval}(P, \text{“}\texttt{CreativeUserName_WithSugarOnTop}\text{”}) = 10241.$$

Motivation for (Minimum-) Perfect Hashing



Short IDs

Replace keys with short unique identifies

$$\textbf{eval}(P, \text{“}\texttt{CreativeUserName_WithSugarOnTop}\text{”}) = 10241.$$

Updatable Retrieval: A hash table without keys

- assume we have MPHF P for S
- can store additional data $f(x) \in [k]$ on $x \in S$ in array of length m in position $\textbf{eval}(P, x)$.
  $\hookrightarrow$ array takes $m\lceil \log_2(k) \rceil$ bits

⚠ Weaker than a normal hash table:

- S is static (values updateable)
- trying to access $f(x)$ for $x \notin S$ gives undefined result
- trying to update $f(x)$ for $x \notin S$ destroys information

Content

1. Retrieval Data Structures
- The Retrieval Problem
- Motivation
- Construction Using Peeling

2. (Minimal-) Perfect Hashing
- The Perfect Hashing Problem
- Motivation: Updatable Retrieval
- Construction using Trial and Error
- Construction using Cuckoo Hashing and Retrieval

Brute Force: Perfect Hashing Using Naive Trial and Error

Karlsruhe Institute of Technology

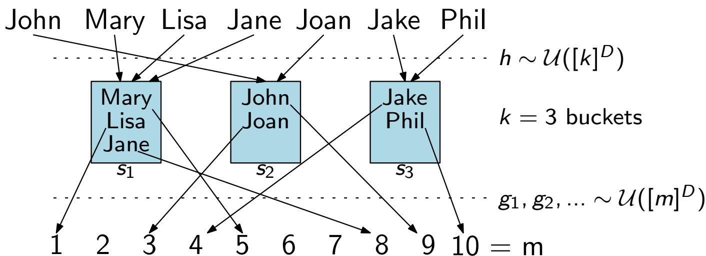
Exercise: What if we played the lottery until we win?

For any $S \subseteq D$ of size n we have $\Pr_{h \sim \mathcal{U}([n]^D)}[h \text{ is injective on } S] = \frac{n!}{n^n}$.

$\hookrightarrow$ Success after trying $\approx \frac{n^n}{n!}$ random hash functions.

$\hookrightarrow$ Need to store seed of $\log_2\left(\frac{n^n}{n!}\right) \approx \log_2(e^n) \approx 1.44n$ bits.

PTHash: Perfect Hashing Using Refined Trial and Error



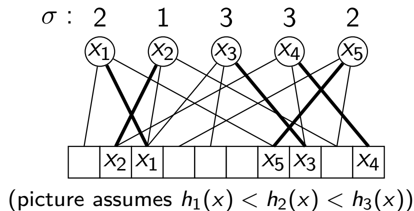
Perfect Hash Function $P = (k, h, (g_i)_{i \in \mathbb{N}}, (s_1, \ldots, s_k))$

- **eval**$(P, x) := g_{s_{h(x)}}(x)$
- $s_1, \ldots, s_k$ are found using trial and error
- *huge design space*

Cuckoo Hashing + Retrieval $\rightarrow$ Perfect Hashing

Cuckoo Hashing (abstract reminder)

Let $S \subseteq D$ of size $n = |S|$ and $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$ where $\frac{n}{m} < c_k^*$ for some *threshold* c_k^*.
With high probability there exists $\sigma(x) \in [k]$ for each $x \in S$ such that $x \mapsto h_{\sigma(x)}(x)$ is injective on S.



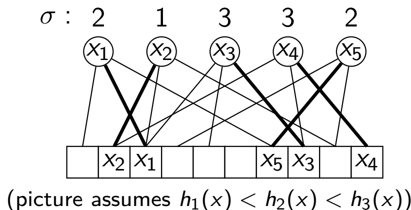
(picture assumes $h_1(x) < h_2(x) < h_3(x)$)

Cuckoo Hashing + Retrieval $\to$ Perfect Hashing

Karlsruhe Institute of Technology

Cuckoo Hashing (abstract reminder)

Let $S \subseteq D$ of size $n = |S|$ and $h_1, \dots, h_k \sim \mathcal{U}([m]^D)$ where $\frac{n}{m} < c_k^*$ for some *threshold* c_k^*.
With high probability there exists $\sigma(x) \in [k]$ for each $x \in S$ such that $x \mapsto h_{\sigma(x)}(x)$ is injective on S.



Perfect Hash Function from Retrieval

- Store $\sigma : S \to [k]$ as retrieval data structure R
- (non-minimal) PHF $P = (R, h_1, \dots, h_k)$ with

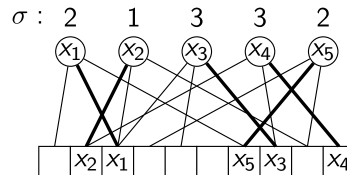
$$\textbf{eval}(P, x) := h_{\textbf{eval}(R,x)}(x).$$

Cuckoo Hashing + Retrieval $\to$ Perfect Hashing

Karlsruhe Institute of Technology

Cuckoo Hashing (abstract reminder)

Let $S \subseteq D$ of size $n = |S|$ and $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$ where $\frac{n}{m} < c_k^*$ for some *threshold* c_k^*.
With high probability there exists $\sigma(x) \in [k]$ for each $x \in S$ such that $x \mapsto h_{\sigma(x)}(x)$ is injective on S.



(picture assumes $h_1(x) < h_2(x) < h_3(x)$)

Perfect Hash Function from Retrieval

- Store $\sigma : S \to [k]$ as retrieval data structure R
- (non-minimal) PHF $P = (R, h_1, \ldots, h_k)$ with

$$\textbf{eval}(P, x) := h_{\textbf{eval}(R,x)}(x).$$

Example with $k = 4$

- need $\frac{n}{m} < c_4^* \approx 0.9768 \rightsquigarrow \varepsilon \approx 0.0238$
- space needed for P is the space for R: $\approx 1.23n\log_2(k) = 2.26n$ bits with the approach from slide 13

Conclusion

KIT Karlsruhe Institute of Technology

Space efficient data structures for special purposes

- (M)PHF for $S \subseteq D$ realises injective function on S, without storing S.
- retrieval data structure for $f : S \to [k]$ can reproduce $f(x)$ for each $x \in S$, without storing S.

Relationships we saw:

Peeling $\to$ Retrieval using $1.23n\lceil\log_2(k)\rceil$ bits

4-ary Cuckoo Hashing + Retrieval $\to$ Perfect Hashing using ≈ 2.26 bits $(\varepsilon = 0.0238)$

Perfect Hashing $\to$ Updatable Retrieval ("hash table without keys")

Remark: There is more...

- Best constructions are more complicated. Not here...
- Active research @ITI Sanders.

Anhang: Mögliche Prüfungsfragen I

Karlsruhe Institute of Technology

- Retrieval Datenstrukturen
  - Was ist der Funktionsumfang einer Retrieval Datenstruktur?
  - Was sind die Vorteile und Nachteile im Vergleich zu einer normalen Hashtabelle?
  - Welche Anwendungen für Retrieval Datenstrukturen haben wir kennengerlernt?
  - Wie lässt sich eine Retrieval Datenstruktur mithilfe des Schälalgorithmus konstruieren? Was sind Konstruktions- und Zugriffszeiten? Was der Speicherverbrauch?
- Perfekte Hashfunktionen
  - Was zeichnet eine gute Perfekte Hashfunktion aus?
  - Wir haben Hashtabellen ohne Schlüssel kennengelernt. Was hat es damit auf sich?
  - Wie kann man perfekte Hashfunktionen mit Trial und Error konstruieren?
  - Wie kann man perfekte Hashfunktionen mittels Cuckoo Hashing und Retrieval konstruieren? Was ist dabei der Speicherverbrauch?