Karlsruhe Institute of Technology

Probability and Computing – Retrieval Data Structures

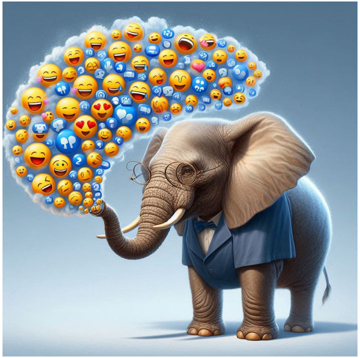
Stefan Walzer | WS 2024/2025

Evaluationsergebnisse

Karlsruhe Institute of Technology

Zusammenfassung der Befragung ($n = 8$)

- Inhalt eher schwer… (3.8 / 5)
- … aber anschaulich (1.5 / 5)
- Aufwand eher hoch… (3.4 / 5)
- … aber sehr lehrreich (1.3 / 5)
- Aufwand Vorbereitung/Nachbereitung
  - 0h-1h ($\times$ 2)
  - 1h-2h ($\times$ 4)
  - 2h-3h ($\times$ 2)
- sehr geeignete Materialien (1.4 / 5)
- insgesamt noch sehr gut (Note 1.4)



Weitere Rückmeldungen oder Verbesserungsvorschläge?

Verbleibende Veranstaltungen & Prüfungstermine

Do	23.1.	reguläre Vorlesung Peeling
Di	28.1.	reguläre Übung
Do	30.1.	Vorlesung Retrieval mit Übung
Do	6.2.	Q&A, weitere Übungsaufgaben, könnte ausfallen
Di	11.2.	~~Übung fällt definitiv aus~~
Do	13.2.	*Gastvorlesung Hans-Peter Lehmann: Perfect Hashing*
	15.2.	*offizielles Ende der Vorlesungszeit*
Mi	26.2.	Prüfungstermine
Do	27.2.	Prüfungstermine
Fr	28.2.	Prüfungstermine
Mi	26.3.	Prüfungstermine
Do	27.3.	Prüfungstermine
Fr	28.3.	Prüfungstermine

Andere Prüfungstermine auf Anfrage vermutlich möglich.

Prüfung (mündlich, 20 Minuten)

- Anmeldung über das Sekretariat:
  - Anja Blancani (blancani@kit.edu)
  - cc an mich (stefan.walzer@kit.edu)
  - Bitte angeben:
    - Vollständiger Name
    - Matrikelnummer
    - Studienfach
    - Version der Prüfungsordnung
- Abmeldung auch über das Sekretariat
- Ort: Stefans Büro (50.34, Raum 209).
- Zeitslots jeweils:
  - 10:00, 10:25, 10:50, 11:15
  - 14:00, 14:25, 14:50, 15:15
- Inhalte von Vorlesung *und Übung*

Content

The Retrieval Problem

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.

$\mathbf{eval}_R(x)$:

- input: $x \in D$
- output: some value in $[k]$
- requirement: $\mathbf{eval}_R(x) = f(x)$ for all $x \in S$

The Retrieval Problem

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.

$\textbf{eval}_R(x)$:

input: $x \in D$

output: some value in $[k]$

requirement: $\textbf{eval}_R(x) = f(x)$ 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 $\textbf{eval}_R$ is $\mathcal{O}(1)$
- ideally running time of **construct** is $\mathcal{O}(n)$

The Retrieval Problem

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.

$\mathbf{eval}_R(x)$:

input: $x \in D$

output: some value in $[k]$

requirement: $\mathbf{eval}_R(x) = f(x)$ for all $x \in S$

The price to pay

- R cannot be used to decide "is $x \in S$?"
- $\mathbf{eval}_R(x)$ 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 $\mathbf{eval}_R$ 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.

$\textbf{eval}_R(x)$:

input: $x \in D$

output: some value in $[k]$

requirement: $\textbf{eval}_R(x) = f(x)$ for all $x \in S$

The price to pay

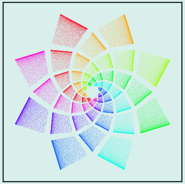
- R cannot be used to decide "is $x \in S$?"
- $\textbf{eval}_R(x)$ is *unspecified* if $x \notin S$.

Goals

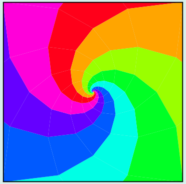
- space requirement of R is $\mathcal{O}(n \log k)$ bits
  - possibly even $n\lceil \log_2(k) \rceil + o(n)$
  - ⚠ naively storing f needs $\Omega(n(\log(k) + \log(|D|)))$
- ideally running time of $\textbf{eval}_R$ is $\mathcal{O}(1)$
- ideally running time of **construct** is $\mathcal{O}(n)$

Intuition

f :



R :

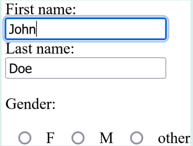


- R is a *continuation* of f
- information about the domain S is lost.

Motivation for Retrieval

Karlsruhe Institute of Technology

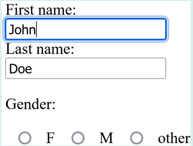
Task: Predict gender based on first name



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

Motivation for Retrieval

Task: Predict gender based on first name



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

Have large data base:

Annotated list of 10000 most common first names.

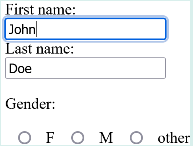
$$f : \{\text{Dave} \mapsto M, \text{Joanna} \mapsto F, \text{Christina} \mapsto F, \ldots\}$$

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

Motivation for Retrieval

Karlsruhe Institute of Technology

Task: Predict gender based on first name



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

Have large data base:

Annotated list of 10000 most common first names.

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

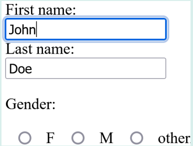
$\approx$ 10 bytes per name, too large to send to client.

Solution using retrieval

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

Motivation for Retrieval

Task: Predict gender based on first name



- 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 $\textbf{eval}_R(\text{firstName})$

Weaknesses:

May guess incorrectly if

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

Static Filters / AMQs

Exercise: Filters from Retrieval

Good retrieval data structures yield good static filters.

Content

Fingerprint-Based Retrieval

x	$f(x)$
Inception	👍
Life of Brian	👎
Avatar	👉
Titanic	👎
The Matrix	👉
Gladiator	👉
Forrest Gump	👍
Interstellar	👎
The Godfather	👍
Jurassic Park	👍
The Lion King	👉
Frozen	👍

Idea 0: Use dictionaries after all

problem	**how to address?**
$(x, f(x))$ too large	

Algorithm eval(x)**:**
  return dict$[x]$

Fingerprint-Based Retrieval

x	$\mathrm{fp}(x)$	$f(x)$
Inception	I	👍
Life of Brian	L	👎
Avatar	A	👉
Titanic	T	👎
The Matrix	T	👉
Gladiator	G	👉
Forrest Gump	F	👍
Interstellar	I	👎
The Godfather	T	👍
Jurassic Park	J	👍
The Lion King	T	👉
Frozen	F	👍

Idea 0: Use dictionaries after all

Idea 1: Store fingerprint-value pairs instead of key-value pairs

- Sample fingerprint hash function $\mathrm{fp} \sim \mathcal{U}([\ell]^D)$ for small ℓ.

problem
$(x, f(x))$ too large

how to address?
fingerprinting: $(\mathrm{fp}(x), f(x))$ is small

Algorithm eval(x)**:**
  return dict[fp(x)]

Fingerprint-Based Retrieval

x	fp(x)	$f(x)$
Inception	**I**	👍
Life of Brian	L	👎
Avatar	A	👉
Titanic	**T**	👎
The Matrix	**T**	👉
Gladiator	G	👉
Forrest Gump	**F**	👍
Interstellar	**I**	👎
The Godfather	**T**	👍
Jurassic Park	J	👍
The Lion King	**T**	👉
Frozen	**F**	👍

Idea 0: Use dictionaries after all

Idea 1: Store fingerprint-value pairs instead of key-value pairs

- Sample fingerprint hash function fp $\sim \mathcal{U}([\ell]^D)$ for small ℓ.

problem	**how to address?**
$(x, f(x))$ too large	fingerprinting: $(\text{fp}(x), f(x))$ is small
fingerprint collisions	

Algorithm eval(x)**:**

  return dict[fp(x)]

Fingerprint-Based Retrieval

Idea 0: Use dictionaries after all

Idea 1: Store fingerprint-value pairs instead of key-value pairs

- Sample fingerprint hash function fp $\sim \mathcal{U}([\ell]^D)$ for small ℓ.

Idea 2: Partition into b small buckets (Example: $b = 3$)

- Sample partition hash function part $\sim \mathcal{U}([b]^D)$.
- Use many dictionaries $\text{dict}_1, \ldots, \text{dict}_b$ of small capacity c (example: $c = 4$).
- Store $\text{fp}(x) \mapsto f(x)$ in $\text{dict}_{\text{part}(x)}$.

x	$\text{fp}(x)$	$f(x)$	$\text{part}(x)$
Gladiator	G	👉	1
The Lion King	T	👉	1
Forrest Gump	F	👍	1
The Matrix	**T**	👉	2
Avatar	A	👉	2
Interstellar	I	👎	2
The Godfather	**T**	👍	2
Inception	I	👍	3
Titanic	T	👎	3
Life of Brian	L	👎	3
Jurassic Park	J	👍	3
Frozen	F	👍	3

Data Structure:

bucket 1	F	G	T	-	👍	👉	👉	-
bucket 2	A	I	-	-	👉	👎	-	-
bucket 3	I	J	L	T	👍	👍	👎	👎

Algorithm eval(x)**:**

$i \leftarrow \text{part}(x)$
return $\text{dict}_i[\text{fp}(x)]$

problem	**how to address?**
$(x, f(x))$ too large	fingerprinting: $(\text{fp}(x), f(x))$ is small
fingerprint collisions	reduced by partitioning,
dict_i **might overflow**	

Fingerprint-Based Retrieval

x	fp(x)	$f(x)$	part(x)
Gladiator	G	👉	1
The Lion King	T	👉	1
Forrest Gump	F	👍	1
The Matrix	**T**	👉	2
Avatar	A	👉	2
Interstellar	I	👎	2
The Godfather	**T**	👍	2
Inception	I	👍	3
Titanic	T	👎	3
Life of Brian	L	👎	3
Jurassic Park	J	👍	3
Frozen	F	👍	3

Idea 0: Use dictionaries after all

Idea 1: Store fingerprint-value pairs instead of key-value pairs

- Sample fingerprint hash function fp $\sim \mathcal{U}([\ell]^D)$ for small ℓ.

Idea 2: Partition into b small buckets (Example: $b = 3$)

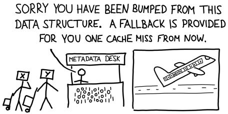
- Sample partition hash function part $\sim \mathcal{U}([b]^D)$.
- Use many dictionaries $\text{dict}_1, \ldots, \text{dict}_b$ of small capacity c (example: $c = 4$).
- Store $\text{fp}(x) \mapsto f(x)$ in $\text{dict}_{\text{part}(x)}$.

Data Structure:

bucket 1	F	G	T	-	👍	👉	👉	-
bucket 2	A	I	-	-	👉	👎	-	-
bucket 3	I	J	L	T	👍	👍	👎	👎

Idea 3: *Bump* inconvenient keys to fallback data structure

Recursively constructed, for small fraction of keys.



problem	**how to address?**
$(x, f(x))$ too large	fingerprinting: $(\text{fp}(x), f(x))$ is small
fingerprint collisions	reduced by partitioning, solved by bumping colliding keys
dict_i **might overflow**	bump excess keys

Algorithm eval(x):

$i \leftarrow \text{part}(x)$
if dict_i.*contains*(fp(x)) **then**
  return $\text{dict}_i[\text{fp}(x)]$
else
  return fallback.**eval**(x)

In expectation: construction time $\mathcal{O}(n)$, query time $\mathcal{O}(1)$, space $\mathcal{O}(\log_2 k)$ bits per key (Müller et al, 2014).

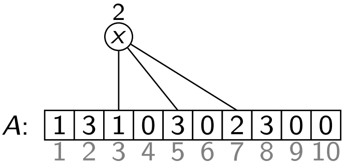
Content

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

KIT Karlsruhe Institute of Technology

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$



$(k = 4)$

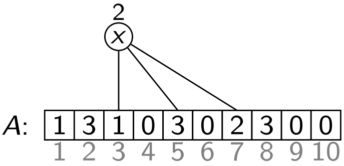
Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \ldots, k-1\}$

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

- $A \in \{0, \ldots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //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 $\mathcal{O}(1)$



$(k = 4)$

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

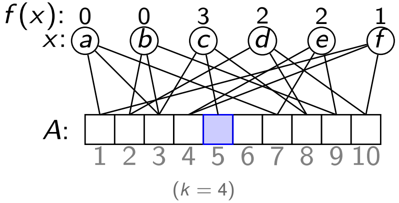
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

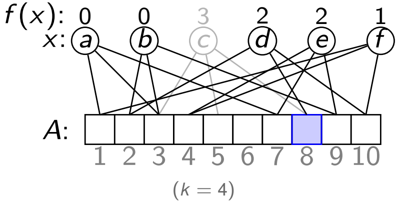
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \ldots, k-1\}$

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

- $A \in \{0, \ldots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\textbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

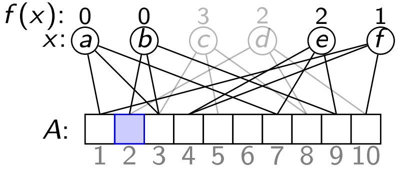
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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



$(k = 4)$

Equations (mod k for $k = 4$)

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

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

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

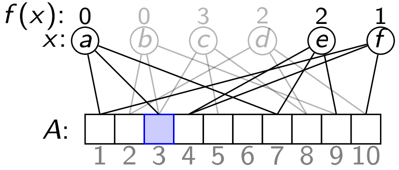
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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 $(\bmod k$ 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: $A[3] := 0 - A[1] - A[7]$

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\textbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

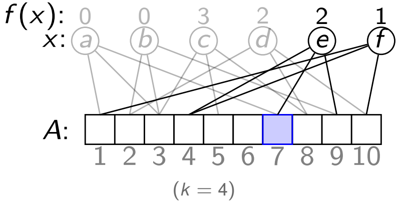
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

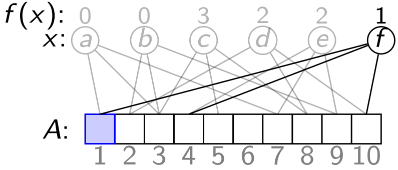
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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



$(k = 4)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\textbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

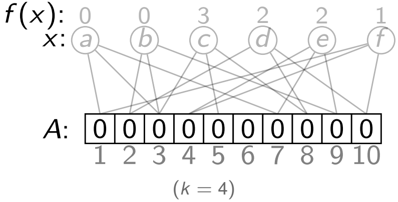
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \ldots, k-1\}$

Karlsruhe Institute of Technology

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

- $A \in \{0, \ldots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

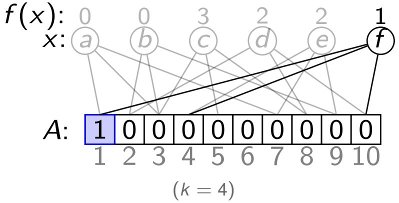
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

Karlsruhe Institute of Technology

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

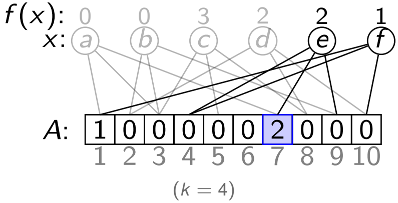
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

Karlsruhe Institute of Technology

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

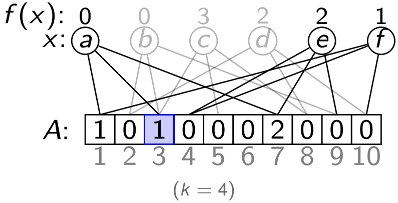
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

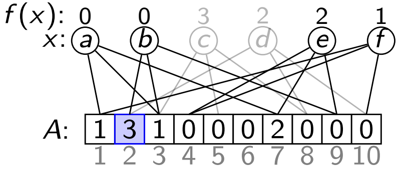
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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



$(k = 4)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^\triangle$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- **eval**$_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

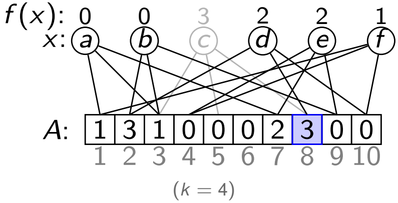
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\mathbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

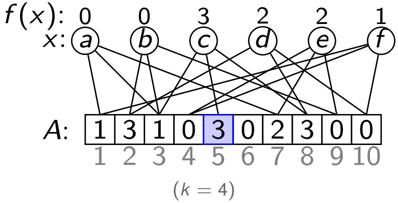
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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

Cuckoo-Style Retrieval using Peeling for $f : S \to \{0, \dots, k-1\}$

Karlsruhe Institute of Technology

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

- $A \in \{0, \dots, k-1\}^m$ is array of cleverly chosen values
- $m = \frac{n}{0.81} = 1.23n$ //0.81 is peeling threshold $c_3^{\triangle}$
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\textbf{eval}_R(x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

Performance

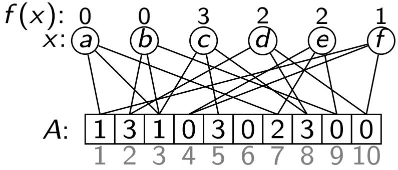
- space $1.23n\lceil \log_2(k) \rceil$ bits
- construct in $\mathcal{O}(n)$
- eval in $\mathcal{O}(1)$

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



$(k = 4)$

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

Cuckoo-Style Retrieval using Linear Algebra over the field $\mathbb{F}_2 = \{0, 1\}$

Cuckoo-style retrieval for $f : S \to \mathbb{F}_2^r$ with $|S| = n$

Pick $m \geq n$. Data structure is pair $R = (h : D \to \mathbb{F}_2^m, \vec{z} \in \mathbb{F}_2^{m \times r})$ such that $h(x)^T \cdot \vec{z} = f(x)$ for all $x \in S$.

$r = 3$
$m = 7$
$n = 5$

$$\underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}}_{} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} \overset{!}{=} f(x)$$

$$h(x) = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Cuckoo-Style Retrieval using Linear Algebra over the field $\mathbb{F}_2 = \{0, 1\}$

Cuckoo-style retrieval for $f : S \to \mathbb{F}_2^r$ with $|S| = n$

Pick $m \geq n$. Data structure is pair $R = (h : D \to \mathbb{F}_2^m, \vec{z} \in \mathbb{F}_2^{m\times r})$ such that $h(x)^T \cdot \vec{z} = f(x)$ for all $x \in S$.

$r = 3$
$m = 7$
$n = 5$

$$\overset{h(x)}{\overset{\shortparallel}{\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}}} \begin{pmatrix} 011 \\ \mathbf{010} \\ 000 \\ 001 \\ \mathbf{000} \\ \mathbf{110} \\ 101 \end{pmatrix} \overset{!}{=} 100$$

Cuckoo-Style Retrieval using Linear Algebra over the field $\mathbb{F}_2 = \{0, 1\}$

Karlsruhe Institute of Technology

Cuckoo-style retrieval for $f : S \to \mathbb{F}_2^r$ with $|S| = n$

Pick $m \geq n$. Data structure is pair $R = (h : D \to \mathbb{F}_2^m, \vec{z} \in \mathbb{F}_2^{m \times r})$ such that $h(x)^T \cdot \vec{z} = f(x)$ for all $x \in S$.

$r = 3$
$m = 7$
$n = 5$

$$\begin{pmatrix} \text{——} h(x_1) \text{——} \\ \text{——} h(x_2) \text{——} \\ \text{——} h(x_3) \text{——} \\ \text{——} h(x_4) \text{——} \\ \text{——} h(x_5) \text{——} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \end{pmatrix}$$

Cuckoo-Style Retrieval using Linear Algebra over the field $\mathbb{F}_2 = \{0, 1\}$

Karlsruhe Institute of Technology

Cuckoo-style retrieval for $f : S \to \mathbb{F}_2^r$ with $|S| = n$

Pick $m \geq n$. Data structure is pair $R = (h : D \to \mathbb{F}_2^m, \vec{z} \in \mathbb{F}_2^{m \times r})$ such that $h(x)^T \cdot \vec{z} = f(x)$ for all $x \in S$.

$r = 3$
$m = 7$
$n = 5$

$$\begin{pmatrix} \text{---}\, h(x_1) \,\text{---} \\ \text{---}\, h(x_2) \,\text{---} \\ \text{---}\, h(x_3) \,\text{---} \\ \text{---}\, h(x_4) \,\text{---} \\ \text{---}\, h(x_5) \,\text{---} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} \overset{!}{=} \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \end{pmatrix}$$

Goals when Choosing h

- i **success probability**: rows of matrix $(h(x))_{x \in S}$ must be linearly independent
- ii **construction time**: linear system should be easy to solve
- iii **query time**: products $h(x) \cdot z$ should be fast to compute
- iv **space**: $\alpha = \frac{n}{m}$ should be close to 1

Cuckoo-Style Retrieval using Linear Algebra over the field $\mathbb{F}_2 = \{0, 1\}$

Karlsruhe Institute of Technology

Cuckoo-style retrieval for $f : S \to \mathbb{F}_2^r$ with $|S| = n$

Pick $m \geq n$. Data structure is pair $R = (h : D \to \mathbb{F}_2^m, \vec{z} \in \mathbb{F}_2^{m \times r})$ such that $h(x)^T \cdot \vec{z} = f(x)$ for all $x \in S$.

$r = 3$
$m = 7$
$n = 5$

$$\begin{pmatrix} \text{——} h(x_1) \text{——} \\ \text{——} h(x_2) \text{——} \\ \text{——} h(x_3) \text{——} \\ \text{——} h(x_4) \text{——} \\ \text{——} h(x_5) \text{——} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} \overset{!}{=} \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \end{pmatrix}$$

Goals when Choosing h

- i **success probability**: rows of matrix $(h(x))_{x \in S}$ must be linearly independent
- ii **construction time**: linear system should be easy to solve
- iii **query time**: products $h(x) \cdot z$ should be fast to compute
- iv **space**: $\alpha = \frac{n}{m}$ should be close to 1

What the peeling-based approach achieves

- i $1 - \mathcal{O}(1/m)$ (success iff peelable)
- ii $\mathcal{O}(n)$ (time to run peeling)
- iii $\mathcal{O}(1)$ (three memory accesses two $\oplus$-additions)
- iv $\alpha = 0.81$ (peeling threshold)

What more could we hope for?

- i -
- ii better cache efficiency
- iii better cache efficiency
- iv $\alpha = 1 - o(1)$

Teaser: “Ribbon Retrieval”

Standard Ribbon Retrieval

Idea: $\boldsymbol{h}(\boldsymbol{x}) =$ “randomly placed block of $\mathcal{O}(\frac{\log n}{1-\alpha})$ random bits”

$$\left(\begin{array}{ccccccccccccccc}
 & & 0 & 1 & 1 & 0 & 0 & 1 & & & & & & & \\
1 & 0 & 1 & 0 & 0 & 1 & & & & & & & & & \\
 & & & & & 0 & 1 & 0 & 1 & 1 & 0 & & & & \\
 & & & & 1 & 0 & 0 & 0 & 1 & 0 & & & & & \\
 & 0 & 1 & 1 & 0 & 1 & 1 & & & & & & & & \\
 & & & & & & & & & 0 & 1 & 0 & 1 & 0 & 1 \\
 & & 0 & 1 & 0 & 1 & 1 & 1 & & & & & & & \\
 & & & & & & \multicolumn{6}{c}{\underbrace{1\quad 1\quad 1\quad 0\quad 0\quad 0}_{\frac{\log n}{1-\alpha}}} & & &
\end{array}\right)
\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \\ z_9 \\ z_{10} \\ z_{11} \\ z_{12} \\ z_{13} \\ z_{14} \\ z_{15} \end{pmatrix}
\overset{!}{=}
\begin{pmatrix} 010 \\ 011 \\ 101 \\ 111 \\ 101 \\ 000 \\ 010 \\ 100 \end{pmatrix}$$

Bumped Ribbon Retrieval

“Bumping” allows reducing the necessary width of the blocks to $\log n$. End result after a lot of effort[a]:

- construction $\mathcal{O}(n \log n)$
  // cache efficient
- query $\mathcal{O}(r)$ // cache efficient
- bits per key: $r + \mathcal{O}(\frac{\log \log n}{\log^2 n})$.

[a] *Fast Succinct Retrieval and Approximate Membership using Ribbon*, Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, Stefan Walzer, 2021.

Summary

$$\begin{array}{lcl} \texttt{John} & \mapsto & ♂ \\ \texttt{Mary} & \mapsto & ♀ \\ \texttt{Lisa} & \mapsto & ♀ \\ \texttt{Carl} & \mapsto & ♂ \\ \texttt{Jane} & \mapsto & ♀ \end{array}$$

Retrieval

- constructed for $f : S \to [k]$

Dictionary / Hash Table

- constructed for $f : S \to [k]$ // $S \subseteq D$

Summary

John $\mapsto$ ♂
Mary $\mapsto$ ♀
Lisa $\mapsto$ ♀
Carl $\mapsto$ ♂
Jane $\mapsto$ ♀

Retrieval

- constructed for $f : S \to [k]$
- goal: $\approx \log_2(k)$ bits per key

Dictionary / Hash Table

- constructed for $f : S \to [k]$ // $S \subseteq D$
- goal: $\approx \log_2(D) + \log_2(k)$ bits per key

Summary

~~John~~ $\mapsto$ ♂
~~Mary~~ $\mapsto$ ♀
~~Lisa~~ $\mapsto$ ♀
~~Carl~~ $\mapsto$ ♂
~~Jane~~ $\mapsto$ ♀

Retrieval

- constructed for $f : S \to [k]$
- goal: $\approx \log_2(k)$ bits per key
- does not store S

$$\textbf{eval}_R(x) = \begin{cases} f(x) & \text{if } x \in S \\ \textit{unspecified} & \text{if } x \notin S \end{cases}$$

Dictionary / Hash Table

- constructed for $f : S \to [k]$ // $S \subseteq D$
- goal: $\approx \log_2(D) + \log_2(k)$ bits per key
- stores S

$$\textbf{lookup}_R(x) = \begin{cases} f(x) & \text{if } x \in S \\ \perp & \text{if } x \notin S \end{cases}$$

Summary

~~John~~ $\mapsto$ ♂
~~Mary~~ $\mapsto$ ♀
~~Lisa~~ $\mapsto$ ♀
~~Carl~~ $\mapsto$ ♂
~~Jane~~ $\mapsto$ ♀

Retrieval

- constructed for $f : S \to [k]$
- goal: $\approx \log_2(k)$ bits per key
- does not store S

$$\textbf{eval}_R(x) = \begin{cases} f(x) & \text{if } x \in S \\ \textit{unspecified} & \text{if } x \notin S \end{cases}$$

- insert & delete not supported

Dictionary / Hash Table

- constructed for $f : S \to [k]$ // $S \subseteq D$
- goal: $\approx \log_2(D) + \log_2(k)$ bits per key
- stores S

$$\textbf{lookup}_R(x) = \begin{cases} f(x) & \text{if } x \in S \\ \perp & \text{if } x \notin S \end{cases}$$

- insert & delete usually supported

Summary

~~John $\mapsto$ ♂~~
~~Mary $\mapsto$ ♀~~
~~Lisa $\mapsto$ ♀~~
~~Carl $\mapsto$ ♂~~
~~Jane $\mapsto$ ♀~~

Retrieval

- constructed for $f : S \to [k]$
- goal: $\approx \log_2(k)$ bits per key
- does not store S

$$\textbf{eval}_R(x) = \begin{cases} f(x) & \text{if } x \in S \\ \textit{unspecified} & \text{if } x \notin S \end{cases}$$

- insert & delete not supported

Dictionary / Hash Table

- constructed for $f : S \to [k]$ // $S \subseteq D$
- goal: $\approx \log_2(D) + \log_2(k)$ bits per key
- stores S

$$\textbf{lookup}_R(x) = \begin{cases} f(x) & \text{if } x \in S \\ \perp & \text{if } x \notin S \end{cases}$$

- insert & delete usually supported

Constructions discussed here

- fingerprint-based approaches
- cuckoo-style retrieval using peeling
- cuckoo-style retrieval using linear algebra with $\mathbb{F}_2$

Remark: There is more...

- compressed retrieval data structures
- learned retrieval data structures
- active research @ITI Sanders!

Anhang: Mögliche Prüfungsfragen

- 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?
- Zu Retrieval Datenstruktur mit dem Fingerprint-Ansatz:
  - Wie ist die Datenstruktur aufgebaut? Wie funktioniert der Query Algorithmus?
  - Was ist „Bumping“ und wieso brauchen wir es?
  - Was sind Konstruktions- und Zugriffszeiten sowie Speicherverbrauch?
- Zu Retrieval Datenstruktur basierend auf dem Schälalgorithmus:
  - Wie ist die Datenstruktur aufgebaut? Wie funktioniert der Query Algorithmus?
  - Was sind Konstruktions- und Zugriffszeiten sowie Speicherverbrauch?
- Zu Retrieval Datenstruktur basierend auf linearer Algebra mit dem Körper $\mathbb{F}_2$:
  - Was ist das allgemeine Schema? Inwiefern passt auch der Ansatz mit dem Schälalgorithmus in dieses Schema?
  - Welche Ziele sollte ich bei der Wahl der Funktion h im Auge behalten?
- Ribbon Retrieval ist nicht prüfungsrelevant.