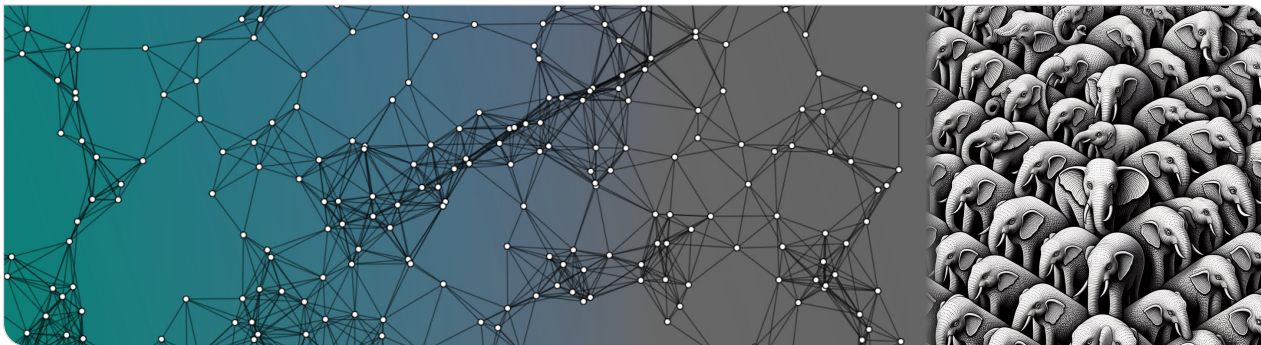


Probability and Computing – Retrieval and Perfect Hashing

Stefan Walzer, Maximilian Katzmann | WS 2023/2024



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

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

The Retrieval Problem

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

construct(f):

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

output: data structure R .

eval(R, x):

input: $R = \mathbf{construct}(f : S \rightarrow [k]), x \in D$

output: some value in $[k]$

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

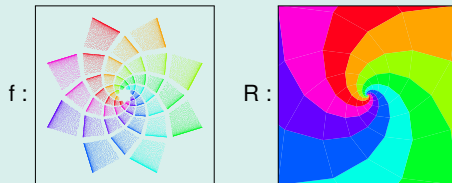
The price to pay

- 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 $n \lceil \log_2(k) \rceil + o(n)$
 - $\triangle!$ 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

Task: Predict gender based on first name

First name:

Last name:

Gender:
 F M other

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

Solution using retrieval

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

Have large data base:

Annotated list of 10000 most common first names.

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

≈ 10 bytes per name, too large to send to client.

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. “Crhristina”, “Inghean”)
 \hookrightarrow would be better to *refrain from guessing*

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

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^Δ
- $A \in [k]^m$ is array of cleverly chosen values
- $h_1, h_2, h_3 \sim \mathcal{U}([m]^D)$ //SUHA
- $\text{eval}(R, x) := (A[h_1(x)] + A[h_2(x)] + A[h_3(x)]) \bmod k$

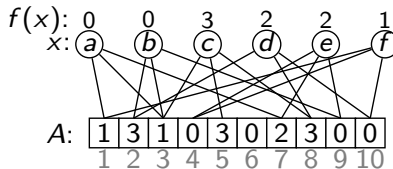
Performance

- 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 $k = 4$)

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

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

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

construct(S):

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

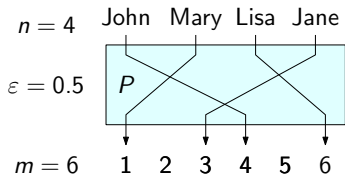
output: data structure P .

eval(P, x):

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

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

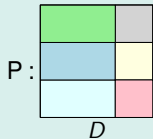
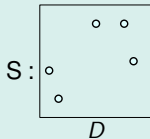
requirement: $x \mapsto \mathbf{eval}(P, x)$ is injective on S



Goals

- ϵ is small // $\epsilon = 0$: *Minimal* perfect hashing
- space requirement of P is $\mathcal{O}(n)$ bits
 - $\approx 1.44n$ bits is necessary and sufficient for $\epsilon = 0$
 - 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



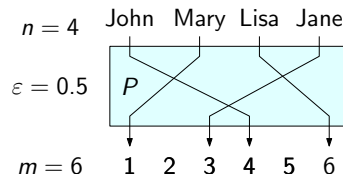
- P is partition of D that separates S
- details about S are lost.
- note: P is “perfect hash function” but need not be random

Motivation for (Minimum-) Perfect Hashing

Short IDs

Replace keys with short unique identifies

$\mathbf{eval}(P, \text{"CreativeUserName_WithSugarOnTop"}) = 10241.$



Updatable Retrieval: A hash table without keys

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

\triangle 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

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

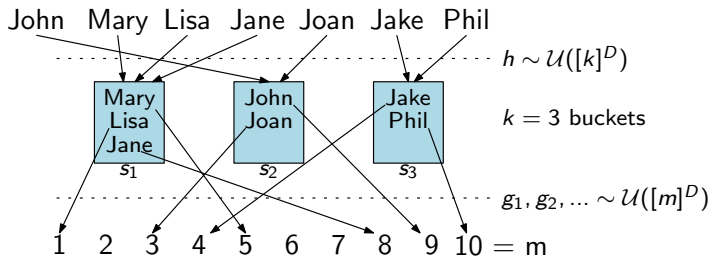
Exercise: What if we played the lottery until we win?

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

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

↪ 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 $P = (k, h, (g_i)_{i \in \mathbb{N}}, (s_1, \dots, s_k))$

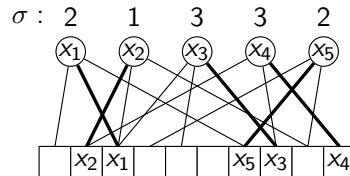
- $\text{eval}(P, x) := g_{s_{h(x)}}(x)$
- s_1, \dots, s_k are found using trial and error
- *huge design space*

Cuckoo Hashing + Retrieval \rightarrow Perfect Hashing

Cuckoo Hashing (abstract reminder)

Let $S \subseteq D$ of size $n = |S|$ and $h_1, \dots, h_k \sim \mathcal{U}([m]^D)$ 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 .



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

Perfect Hash Function from Retrieval

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

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

Example with $k = 4$

- need $\frac{n}{m} < c_4^* \approx 0.9768 \rightsquigarrow \epsilon \approx 0.0238$
- space needed for P is the space for R :
 $\approx 1.23n \log_2(k) = 2.26n$ bits
with the approach from slide 7

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 $f : S \rightarrow [k]$ can reproduce $f(x)$ for each $x \in S$, without storing S .

Relationships we saw:

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

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

Perfect Hashing \rightarrow Updatable Retrieval (“hash table without keys”)

Remark: There is more...

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

- 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 kennengelernt?
 - 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?