Perfekte Hashfunktionen
–
so kompakt wie es geht

Yaojia Hou & Christian Schwarz

Fachgebiet Komplexitätstheorie,
Institut für Theoretische Informatik,
Technische Universität Ilmenau

18. November 2021

Das Hashing-Problem

Gegeben: U .. Universum,
$S \subseteq U$ .. Schlüssel
$m = (1 + \epsilon)|S|$

Konstruiere: Abbildung $H : U \to [m]$, mit $H|S$ injektiv
Datenstruktur DS zum Speichern von H

Ziel: Speicherplatz für DS klein
Auswerteszeit für H konstant
auch für große $|S|$

Roadmap

Einführung

Konstruktion

Partitionierung (große Schlüsselmengen)

Auswertung & Fazit

Einführung

Konstruktion

Partitionierung (große Schlüsselmengen)

Auswertung & Fazit

Konstruktion

für jeden Schlüssel $x \in S$, berechne k Zahlen $h_x^1, \ldots, h_x^k \in [m]$

$B\ x = (h_x^1, \ldots, h_x^k)$

gesucht

Abbildung $\sigma : S \longrightarrow [k]$ so, dass

$$S \longrightarrow [m] : x \mapsto h_x^{\sigma(x)} \text{ injektiv ist}$$

Konstruktion ('2)

Wähle abhängig von ϵ einen von zwei Algorithmen:

$m = (1 + \epsilon)n$ mit $\epsilon \in \{1,\ 0.23,\ 0.04,\ 0\}$

$\epsilon = 0.23 \longrightarrow$ *peeling* eines Hypergraphen — Botelho et al. [BPZ13]

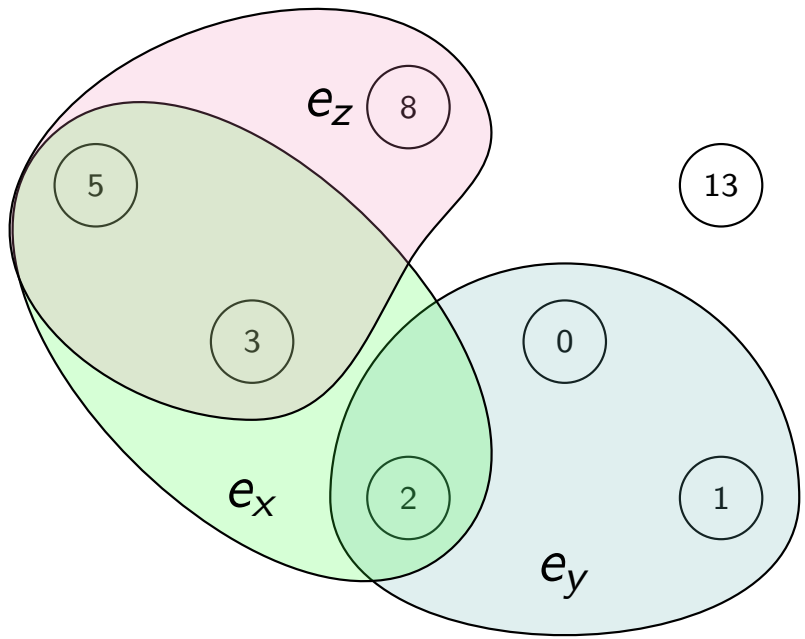
sonst $\longrightarrow$ *Cuckoo Insertion*
$+$ Lösen eines LGS — VL "Hashing"

$\epsilon = 0 \longrightarrow$ arbeite mit $m = 1.04n$ & konstruiere Rang DS

Peeling

geg.: für jeden Schlüssel $x \in S$, k Zahlen $h_x^1, \ldots, h_x^k \in [m]$

die Tupel $e_x := (h_x^1, \ldots, h_x^k)$ bilden Kanten eines (k-uniformen) Hypergraphen H



Peeling ('2)

Ergebnis

Aufzählung $\underline{e} = (e_1, e_2, \ldots)$ der Kanten von H so, dass:

$$e_{i+1} \text{ hat Ecke vom Grad 1 in } H - \{e_1, \ldots, e_i\}$$

kann scheitern, falls H 'Kreise' hat

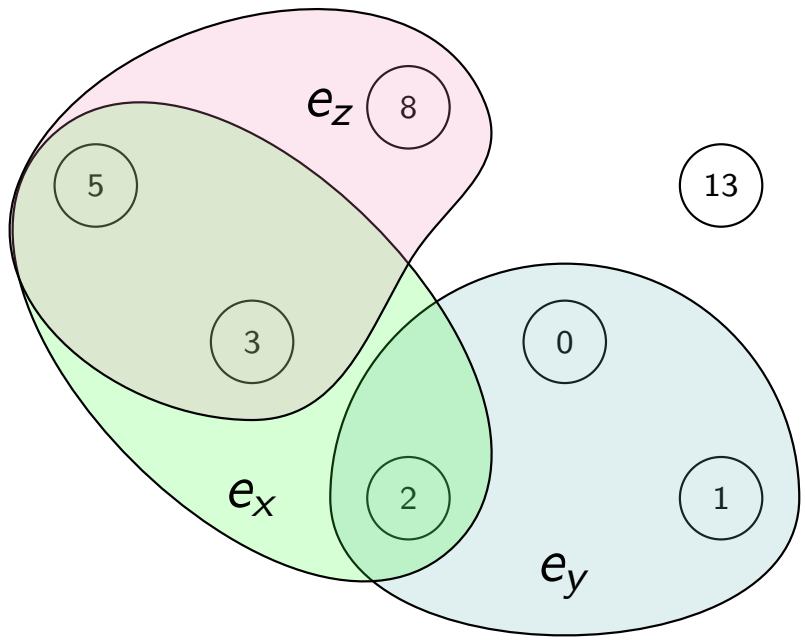
Algorithmus:

entferne sukzessive Kanten e mit einer Ecke v vom Grad 1

Peeling (Beispiel)

Algorithmus:

entferne sukzessive Kanten e mit einer Ecke v vom Grad 1



$\longrightarrow \underline{e} = (e_z, e_x, e_y)$ $\qquad\qquad$ $\underline{v} = (8, 3, 1)$

Darstellung von σ

$\sigma(x)$ – Auswahl eines Hashwertes $h_x^1, \ldots, h_x^k$

peeling liefert Aufzählung $\underline{e}$ und Folge $\underline{v}$

$\underline{v}_i$ pw. verschieden
'$\underline{v}_i \in \underline{e}_i$'

für $x \in S$ sei j der Index von $e_x = (h_x^1, \ldots, h_x^k)$ in $\underline{e}$
wähle $\sigma(x)$ so, dass $\underline{v}_j = h_x^{\sigma(x)}$

Problem: Wie ist σ zu speichern?

Darstellung von σ ('2)

Array $\mathrm{sigma}[m]$ von Werten aus $\{0, .., k\}$

$\sim$ Ecken-Label in H

Idee

$$\sigma(x) - 1 \stackrel{!}{=} \left(\mathrm{sigma}[h_x^1] + \cdots + \mathrm{sigma}[h_x^k]\right) \bmod k$$

$\sigma(x) \in \{1, \ldots, k\}$

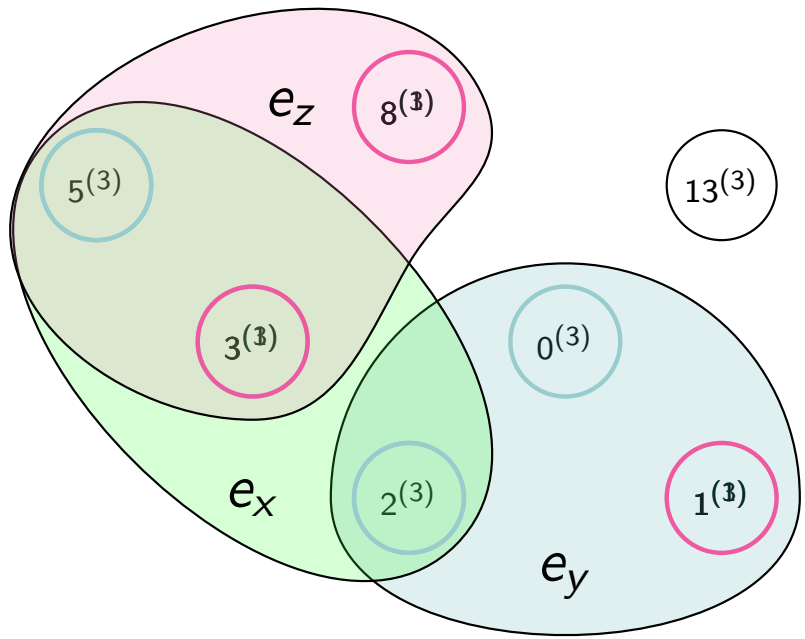
initial: alle Einträge $= k$

durchlaufe $\underline{e}$ in *umgekehrter Reihenfolge* $(j = n..1)$
und passe $\mathrm{sigma}[\underline{v}_j]$ an

Darstellung von σ (Beispiel)

$\underline{e} = (e_z, e_x, e_y)$ und $\underline{v} = (8, 3, 1)$

$$\text{sigma}[3, 1, 3, 1, \ldots]$$



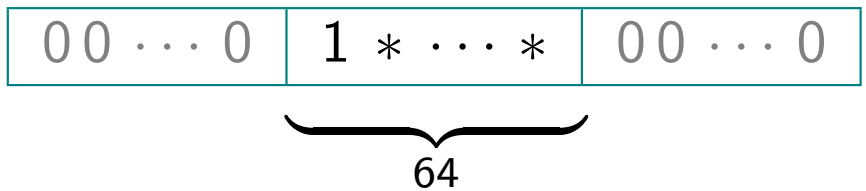
Ecken in e seien der Größe nach sortiert,
also $h_x^1 < \ldots < h_x^3$

Variante aus der VL (Übersicht)

geg.: für jeden Schlüssel $x \in S$, k Zahlen $h_x^1, \ldots, h_x^k \in [m]$

berechne σ mittels *Cuckoo Hashing*

konstruiere Matrix $A \in \{0,1\}^{S \times [m]}$ mit Zeilen der Form



$* \in \{0,1\}$

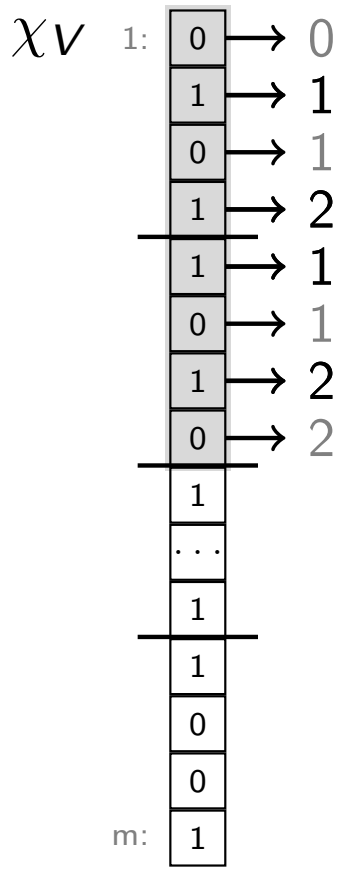
löse das LGS

$$A\,\overline{z} = (\sigma(x))_{x \in S} \qquad \overline{z} \in \{0,1\}^{m \times \lceil \log k \rceil}$$

Rang Datenstruktur

Für Menge $V \subseteq [m]$ sei $\mathrm{rank}_V(x) := |\{v \in V : v \leq x\}|$

Idee: speichere Char. Vektor von V



$\longrightarrow$ ersetze 0101 und 1010
durch 'Abbildung' $[1, 1, 2, 2]$

teile $\{0000, \ldots, 1111\}$ in Klassen auf
$\longrightarrow$ 8 Klassen
$\longrightarrow$ 3 Bit (statt 4)
$\longrightarrow$ $0.75m$ *Bit* insgesamt

Einführung

Konstruktion

Partitionierung (große Schlüsselmengen)

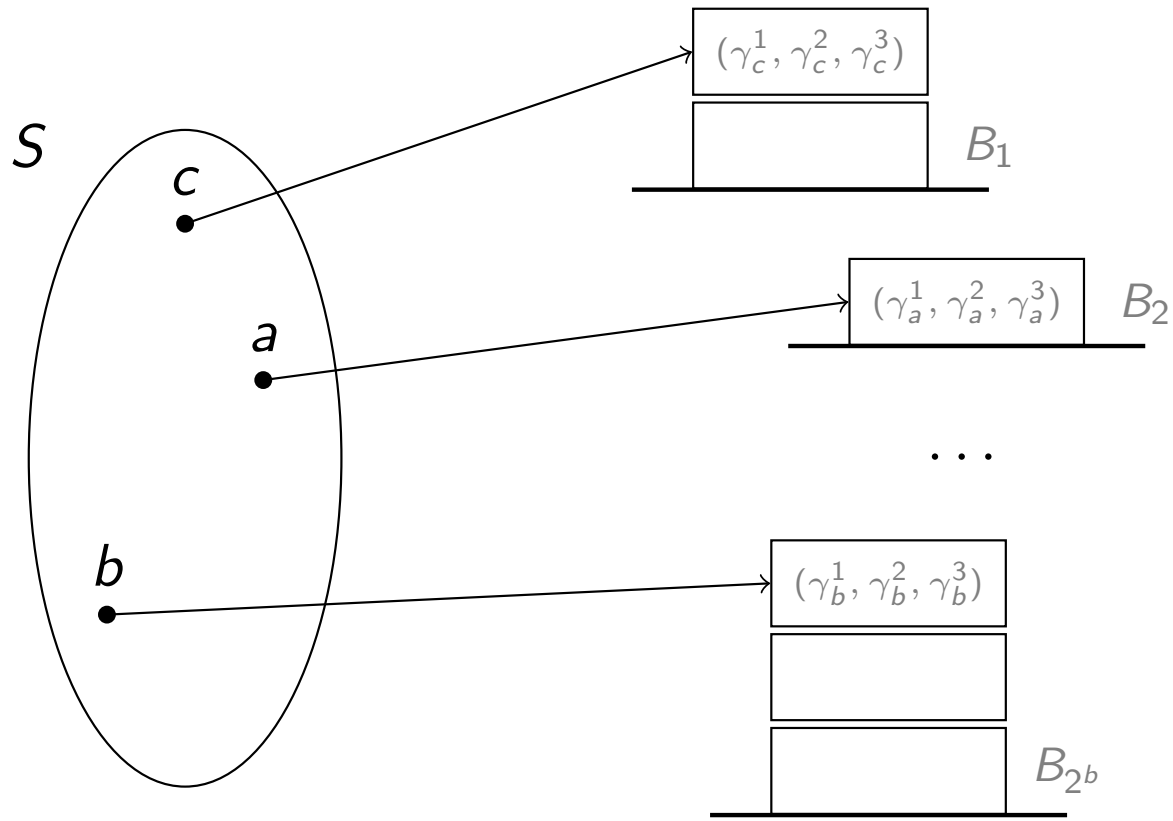
Auswertung & Fazit

große Schlüsselmengen

Teile S auf in 2^b *buckets* $B_1, B_2, \ldots$

Berechne für jedes $x \in S$ Werte $\gamma_x^1, \gamma_x^2, \gamma_x^3$ aus $[2^{16}]$

(Identität der Schlüssel wird danach 'vergessen')



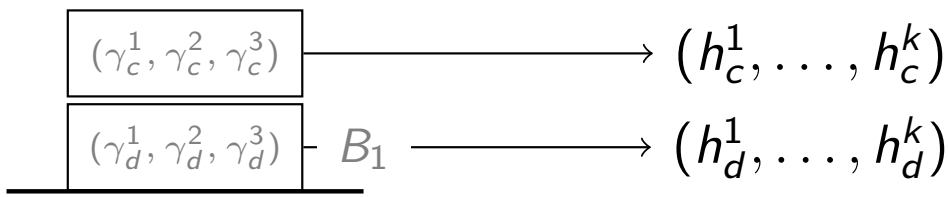
große Schlüsselmengen ('2)

Konstruiere $(\mathrm{M})\mathrm{PHF}$ H_i für jeden bucket B_i

$(n = |B_i|, m = (1+\epsilon)\max|B_j|)$

$h_x^1, \ldots, h_x^k$ werden aus den γ_x^i berechnet

(benötigt große Tabellen!)



verschiebe $H_i(x)$ um $(i-1) \times m$ bzw. $\sum_{j<i} |B_j|$

Einführung

Konstruktion

Partitionierung (große Schlüsselmengen)

Auswertung & Fazit

Übersicht der Ergebnisse

$n = 3$Mio. Schlüssel (Zeichenketten der Länge 10)

Konfiguration		Speicherplatz [Bit/Schlüssel]	Auswertezeit
m=1.23n	EM-T	1.98	24μs
	RAM	1.97	30ns
m=1.04n	EM-T	2.09	33μs
	RAM	—	—
m=n	EM-T	2.98	35μs
	RAM	—	—
	[BPZ13]	2.67	$\approx$ 30ns

Übersicht der Ergebnisse ('2)

$n = 3$Mio. Schlüssel (Zeichenketten der Länge 10)

Konfiguration		Speicherplatz [Bit/Schlüssel]	Auswertezeit
m=1.23n	EM-T	1.98	24μs
	EM$_+$T	14.56	1μs
m=1.04n	EM-T	2.09	33μs
	EM$_+$T	18.87	2μs
m=n	EM-T	2.98	35μs
	EM$_+$T	19.75	2μs

Übersicht der Ergebnisse ('3)

Schlüssel: Wörter mit 20..40 Buchstaben

Konfiguration		Speicherplatz		
n	b	m=1.23n	m=1.04n	m=n
1M	4	2.03	2.09	2.98
100M	11	2.00	2.11	2.99
1.75G	15	—	—	—

Fazit

Perfekte Hashfunktionen

Hinsichtlich *Speicherplatz* und *Auswertezeit*:
Algo. von Botelho et Al. besser (1.97 B/S, $< 0.1\mu$s)
aber: 23% größerer Bildbereich

Algo. aus VL nur 4..5%, dafür 2.1 B/S und $\approx 30\mu$s
$\longrightarrow$ +0.13 Bit/Schlüssel
$\longrightarrow$ aber! 1000$\times$ langsamer

Fazit ('2)

Minimal Perfekte Hashfunktionen

Algo. von Botelho et Al.: 2.62 B/S und $< 1\mu$s (hier nicht getestet)

Algo. aus VL: 2.98 B/S und $\approx 30\mu$s
$\longrightarrow +0.36$ B/S
$\longrightarrow \approx 100\times$ langsamer

Fazit ('3)

zwei Algorithmen implementiert und verglichen

- Speicherplatz: ähnlich (< 0.4 B/S Differenz)
- Auswertezeit: Algo. aus VL zu langsam

- Problem: nur für $n < 100\,000$ ist $A\,\overline{z} = (\sigma(x))_x$ lösbar

  bessere Ergebnisse bei anderer Wahl von A?

Es geht noch besser .. (RecSplit / PTHash)