Karlsruher Institut für Technologie

PaCHash: Packed and Compressed Hash Tables

Vorlesung Algorithm Engineering

Hans-Peter Lehmann | 14. Juni 2022

Bitvektoren

- $rank_1(x)$: Anzahl der 1en bis zur Position x
- $select_1(x)$: Position der x-ten 1
- Query-Zeit $O(1)$ mit Platzverbrauch $o(n)$
- Implementierungen mit nur 3% Overhead [8]

0 1 1 0 1 1 1 0 0 1

Bitvektoren

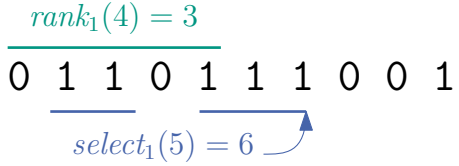
- $rank_1(x)$: Anzahl der 1en bis zur Position x
- $select_1(x)$: Position der x-ten 1
- Query-Zeit $O(1)$ mit Platzverbrauch $o(n)$
- Implementierungen mit nur 3% Overhead [8]

$$rank_1(4) = 3$$

0 1 1 0 1 1 1 0 0 1

Bitvektoren

- $rank_1(x)$: Anzahl der 1en bis zur Position x
- $select_1(x)$: Position der x-ten 1
- Query-Zeit $O(1)$ mit Platzverbrauch $o(n)$
- Implementierungen mit nur 3% Overhead [8]



Elias-Fano Coding [3, 6]

- Kompakte Repräsentation einer monoton wachsenden Folge von n Ganzzahlen $\leq U$

i	$\texttt{arr}[i]$						L[i]	H
0	1	0	0	0	0	1		
1	4	0	0	1	0	0		
2	9	0	1	0	0	1		
3	9	0	1	0	0	1		
4	11	0	1	0	1	1		
...								

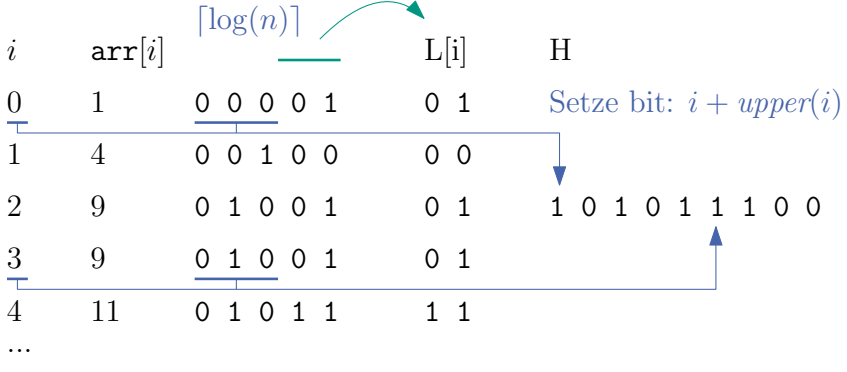
Elias-Fano Coding [3, 6]

- Kompakte Repräsentation einer monoton wachsenden Folge von n Ganzzahlen $\leq U$

i	$\texttt{arr}[i]$		L[i]	H
0	1	0 0 0 0 1	0 1	
1	4	0 0 1 0 0	0 0	
2	9	0 1 0 0 1	0 1	
3	9	0 1 0 0 1	0 1	
4	11	0 1 0 1 1	1 1	
...				

Elias-Fano Coding [3, 6]

- Kompakte Repräsentation einer monoton wachsenden Folge von n Ganzzahlen $\leq U$



Elias-Fano Coding: Zugriff

- Zugriff in $O(1)$ mit $select_1$

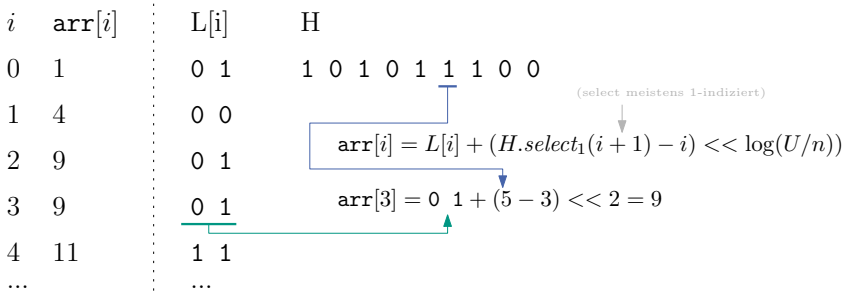
i	$\texttt{arr}[i]$	L[i]	H
0	1	0 1	1 0 1 0 1 1 1 0 0
1	4	0 0	
2	9	0 1	
3	9	0 1	
4	11	1 1	
...		...	

(select meistens 1-indiziert)

$$\texttt{arr}[i] = L[i] + (H.select_1(i+1) - i) << \log(U/n))$$

Elias-Fano Coding: Zugriff

- Zugriff in $O(1)$ mit $select_1$



Elias-Fano Coding: Predecessor Query

- *predecessor*$(x) = \max y \in \texttt{arr} : y \leq x$ in erwarteter Zeit $O(\log(U/n))$

i	$\texttt{arr}[i]$	L[i]	H
0	1	0 1	1 0 1 0 1 1 1 0 0
1	4	0 0	
2	9	0 1	
3	9	0 1	
4	11	1 1	
...		...	

$predecessor(x)$: binäre Suche zwischen
$select_0(upper(x))..select_0(upper(x)+1)$

Elias-Fano Coding: Predecessor Query

- *predecessor*$(x) = \max y \in \texttt{arr} : y \leq x$ in erwarteter Zeit $O(\log(U/n))$

i	arr$[i]$	L[i]
0	1	0 1
1	4	0 0
2	9	0 1
3	9	0 1
4	11	1 1
...		...

H: 1 0 1 0 1 1 1 0 0

predecessor(x) : binäre Suche zwischen $select_0(upper(x))..select_0(upper(x)+1)$

$upper(10) =$ 0 1 0

$predecessor(10) \in select_0(2)..select_0(3)$

Elias-Fano Coding: Speicherplatz

Naiv:

- $n \cdot \log(U)$

Elias-Fano:

- Array L: $n \cdot \log(U/n)$
- Bitvektor H: $2n + 1$
- Select-Datenstruktur: $o(2n + 1)$
- $\rightarrow n \cdot (2 + \log(U/n) + o(1))$

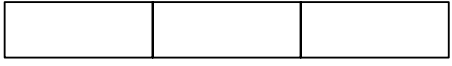
PaCHash: Motivation

	Cuckoo [13]	Interne HT mit Pointern	Separator Hashing [10]	Chaining [2]
Zugriff in 1 I/O	✗	✓	✓	✗
Objekte variabler Größe	✗	✓	✗	✓
Platzeffizient extern	✗	✓	✗	✓
Platzeffizient intern	✓	✗	✓	✓

PaCHash: Motivation

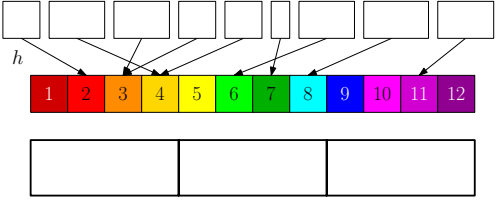
	Cuckoo [13]	Interne HT mit Pointern	Separator Hashing [10]	Chaining [2]	**PaCHash** [9]
Zugriff in 1 I/O	✗	✓	✓	✗	✓*
Objekte variabler Größe	✗	✓	✗	✓	✓
Platzeffizient extern	✗	✓	✗	✓	✓
Platzeffizient intern	✓	✗	✓	✓	✓

PaCHash [9]



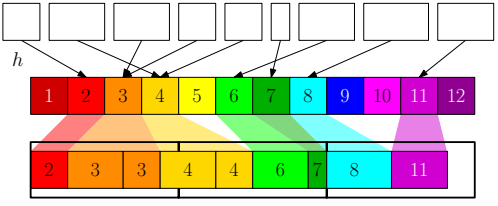
PaCHash [9]

- Objekte in *a* Bins/Block hashen



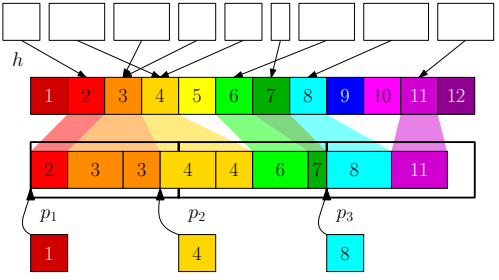
PaCHash [9]

- Objekte in *a* Bins/Block hashen
- Extern: Objekte gepackt, sortiert nach Bins



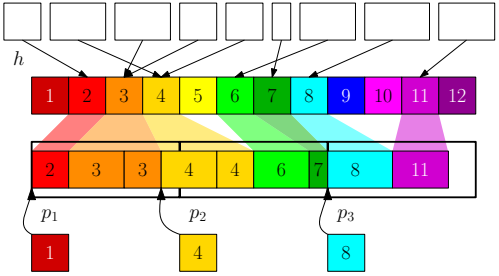
PaCHash [9]

- Objekte in *a* Bins/Block hashen
- Extern: Objekte gepackt, sortiert nach Bins
- Intern: Index speichert ersten Bin pro Block



PaCHash: Analyse

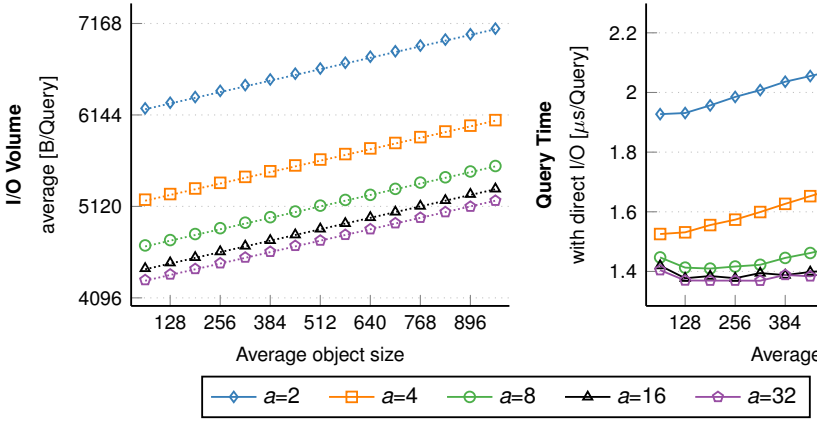
- Array p mit Elias-Fano codiert
  - m Blöcke, am Bins
  - $\log(a)$ bit direkt in L, Rest in H
  - $\rightarrow 2 + \log(a) + o(1)$ bit/Block
- Abrufen von Objekt x
  - Konstanter Lookup in Elias-Fano (Predecessor Query)
  - 1 I/O Operation mit $1 + |x|/B + 1/a$ konsekutiven Blöcken
- Extern fast 100% platzeffizient
  - Je nach Codierung der Objekte vielleicht zusätzlich $\log(B)$ bit Offset pro Block



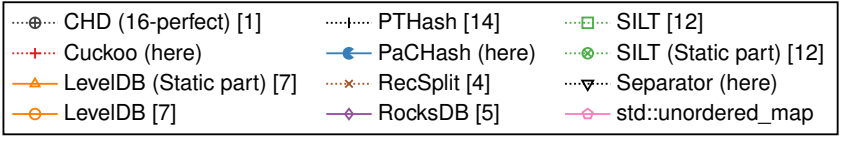
PaCHash: Engineering

- I/O
  - `io_uring` aus Linux 5.1
  - Queue depth: Latenz vs Throughput
- Block-Layout mit Tabelle
  - Schneller zu scannen als Objekte selbst
  - Offsets als Tabellen-Eintrag statt Größe
  - Tabelle am Ende des Blocks für lineares Schreiben
- Hashing: fastrange [11] statt Modulo
  - `r % am` vs `(uint64_t)(((__uint128_t)r * (__uint128_t)am) >> 64)`

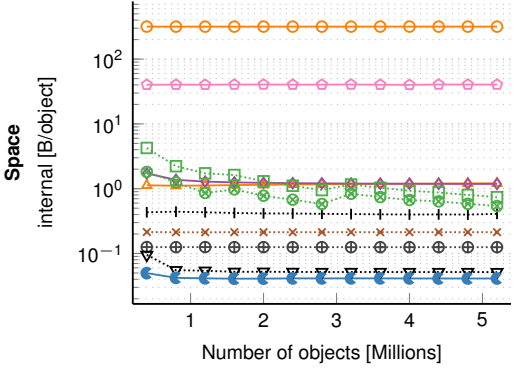
Evaluation: Parameter *a*

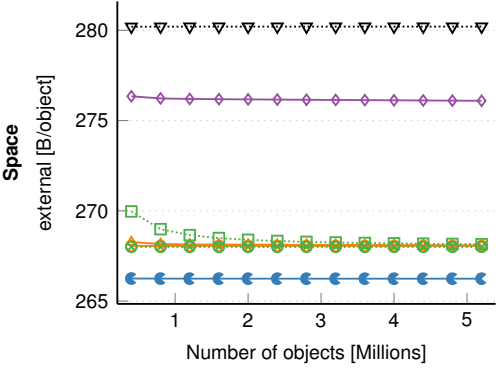


Evaluation: Competitors

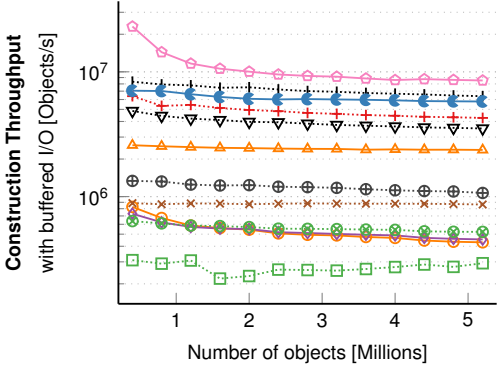


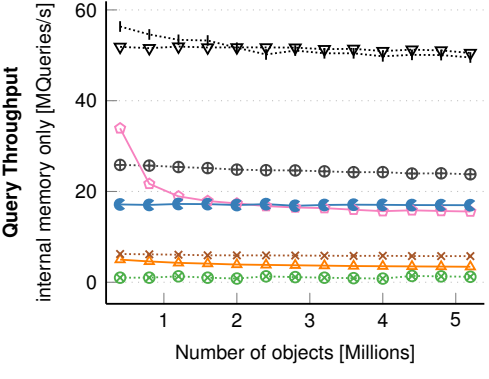
Evaluation: Competitors





Evaluation: Competitors





Quellen I

Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger.
Hash, displace, and compress.
In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 682–693. Springer, 2009.
doi:10.1007/978-3-642-04128-0_61.

Biplob K. Debnath, Sudipta Sengupta, and Jin Li.
Skimpystash: RAM space skimpy key-value store on flash-based storage.
In *SIGMOD Conference*, pages 25–36. ACM, 2011.
doi:10.1145/1989323.1989327.

Peter Elias.
Efficient storage and retrieval by content and address of static files.
J. ACM, 21(2):246–260, 1974.
doi:10.1145/321812.321820.

Quellen II


Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna.
Recsplit: Minimal perfect hashing via recursive splitting.
In *ALENEX*, pages 175–185. SIAM, 2020.
doi:10.1137/1.9781611976007.14.


Facebook.
RocksDB. a persistent key-value store for fast storage environments.
https://rocksdb.org, 2021.


Robert Mario Fano.
On the number of bits required to implement an associative memory.
Technical report, MIT, Computer Structures Group, 1971.
Project MAC, Memorandum 61".

Quellen III


Google.
LevelDB is a fast key-value storage library written at google.
https://github.com/google/leveldb, 2021.


Florian Kurpicz.
Engineering compact data structures for rank and select queries on bit vectors.
CoRR, abs/2206.01149, 2022.


Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders.
Pachash: Packed and compressed hash tables, 2022.
URL: https://arxiv.org/abs/2205.04745, doi:10.48550/ARXIV.2205.04745.

Quellen IV


Per-Åke Larson and Ajay Kajla.
File organization: Implementation of a method guaranteeing retrieval in one access.
Commun. ACM, 27(7):670–677, 1984.
doi:10.1145/358105.358193.


Daniel Lemire.
Fast random integer generation in an interval.
ACM Trans. Model. Comput. Simul., 29(1), jan 2019.
doi:10.1145/3230636.


Hyeontaek Lim, Bin Fan, David G. Andersen, and Michael Kaminsky.
SILT: a memory-efficient, high-performance key-value store.
In *SOSP*, pages 1–13. ACM, 2011.
doi:10.1145/2043556.2043558.

Quellen V


Rasmus Pagh and Flemming Friche Rodler.
Cuckoo hashing.
J. Algorithms, 51(2):122–144, 2004.
doi:10.1016/j.jalgor.2003.12.002.


Giulio Ermanno Pibiri and Roberto Trani.
Pthash: Revisiting FCH minimal perfect hashing.
In *SIGIR*, pages 1339–1348. ACM, 2021.
doi:10.1145/3404835.3462849.