Karlsruhe Institute of Technology

Algorithmen II

Peter Sanders

Übungen:

Moritz Laupichler, Nikolai Maas

Institut für Theoretische Informatik

Web:

`http://algo2.iti.kit.edu/AlgorithmenII_WS23.php`

6 Randomisierte Algorithmen

verwende Zufall(sbits) zur Beschleunigung/Vereinfachung von Algorithmen

Las Vegas: Ergebnis immer korrekt, Laufzeit ist Zufallsvariable.

Schon gesehen:

- quicksort
- hashing

Monte Carlo: Ergebnis mit bestimmter Wahrscheinlichkeit p inkorrekt.

k-fache Wiederholung macht Fehlschlagswahrscheinlichkeit exponentiell klein (p^k).

Mehr in der Vorlesung Randomisierte Algorithmen von Thomas Worsch

6.1 Sortieren – Ergebnisüberprüfung (Checking)

Permutationseigenschaft (Sortiertheit: trivial.)

$\langle e_1, \ldots, e_n \rangle$ ist Permutation von $\langle e'_1, \ldots, e'_n \rangle$ gdw.

$$q(z) := \prod_{i=1}^{n} (z - \text{field}(\text{key}(e_i))) - \prod_{i=1}^{n} (z - \text{field}(\text{key}(e'_i))) = 0,$$

$\mathbb{F}$ sei Körper, field : Key $\to \mathbb{F}$ sei injektiv.

Beobachtung: q hat höchstens n Nullstellen.

Auswertung an zufälliger Stelle $x \in \mathbb{F}$.

$$\mathbb{P}[q \neq 0 \wedge q(x) = 0] \leq \frac{n}{|\mathbb{F}|}$$

Monte Carlo-Algorithmus, Linearzeit.

Frage: Welchen Körper $\mathbb{F}$ nehmen wir?

Sort Checking II – mit Lorenz Hübschle-Schneider

Ist Folge E eine Permutation von Folge E'?

Sei h zufällige Hash-Funktion mit Wertebereich $0..U-1$,

$h(S) := \sum_{e \in S} h(e)$

Checker: return $h(E) = h(E')$

Sort Checking II – mit Lorenz Hübschle-Schneider

Ist Folge E eine Permutation von Folge E'?

Sei h zufällige Hash-Funktion mit Wertebereich $0..U-1$,

$h(S) := \sum_{e \in S} h(e)$

Checker: return $h(E) = h(E')$

Korrekt falls $E = E'$.

Fall $E \neq E'$: Wir zeigen $\mathbb{P}[h(E) = h(E')] \leq \frac{1}{U}$

Sei e ein Element, das $k\times$ in E vorkommt und $k' \neq k\times$ in E'.

$$h(E) = h(E') \Leftrightarrow h(E \setminus e) + kh(e) = h(E' \setminus e) + k'h(e)$$

$$\Leftrightarrow h(e) = \frac{h(E' \setminus e) - h(E \setminus e)}{k - k'} =: x$$

$\mathbb{P}[h(e) = x] \leq \frac{1}{U}$ weil x unabhängig von $h(e)$ ■

6.2 Hashing II

Perfektes Hashing

Idee: mache h injektiv.

Braucht $\geq 1.44n$ bits Platz !

Neuere Ergebnisse (zuletzt auch aus unserer AG) zeigen, dass es tatsächlich praktikable Verfahren gibt, die dieser Schranke sehr nahe kommen (statisch). Mehr in der Übung und der Vorlesung Algorithm Engineering.

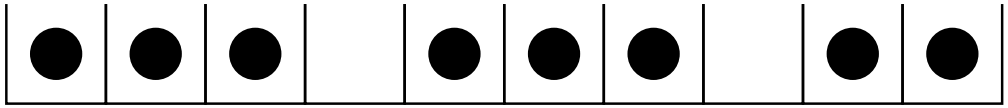
Here: Fast Space Efficient Hashing

Represent a set of n elements (with associated information) using space $(1+\varepsilon)n$.

Support operations insert, delete, lookup, (doall) efficiently.

Assume a truly random hash function h

([Dietzfelbinger, Weidling 2005] shows that this is justified.)



Related Work

Linear probing: $E[T_{\text{find}}] \approx \frac{1}{2\varepsilon^2}$

Uniform hashing: $E[T_{\text{find}}] \approx \frac{1}{\varepsilon}$

Cuckoo Hashing

[Pagh Rodler 01]

Table of size $(2+\varepsilon)n$.

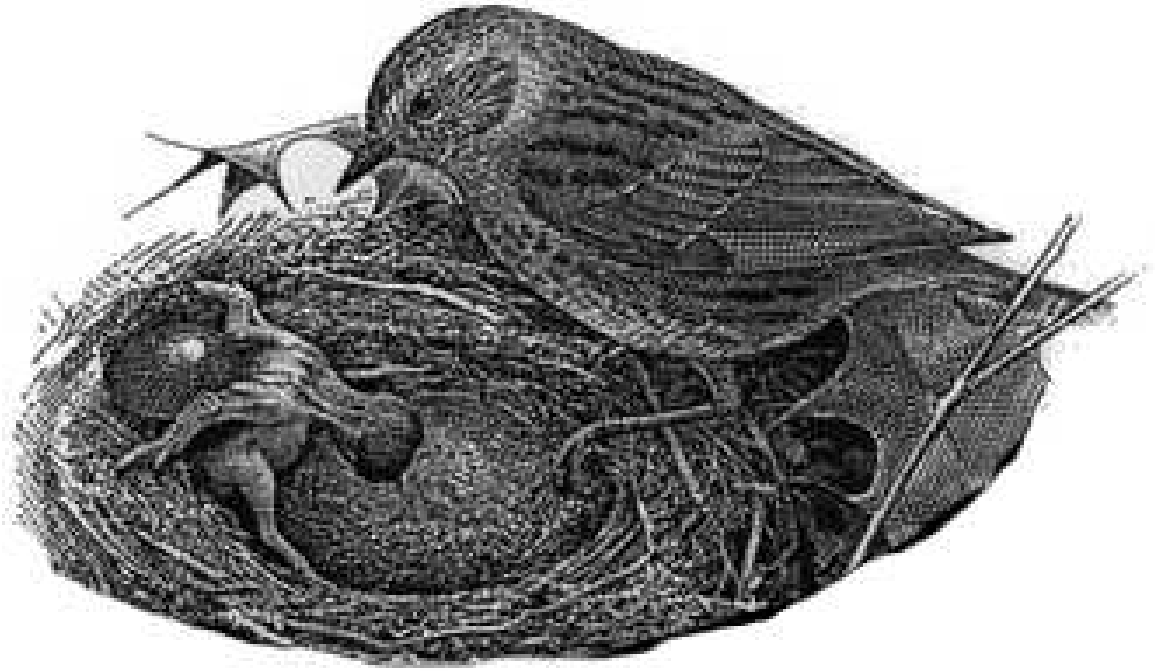
Two choices for each element.

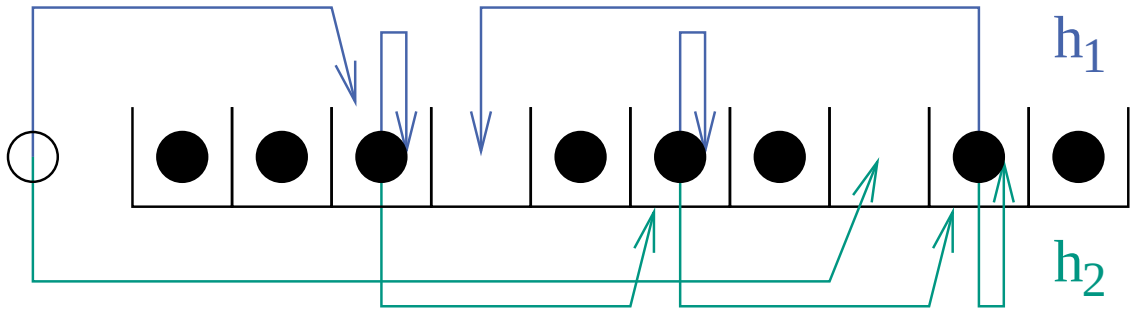
Insert moves elements;

rebuild if necessary.

Very fast lookup and delete.

Expected constant insertion time.





Cuckoo Hashing – Rebuilds

When needed ?

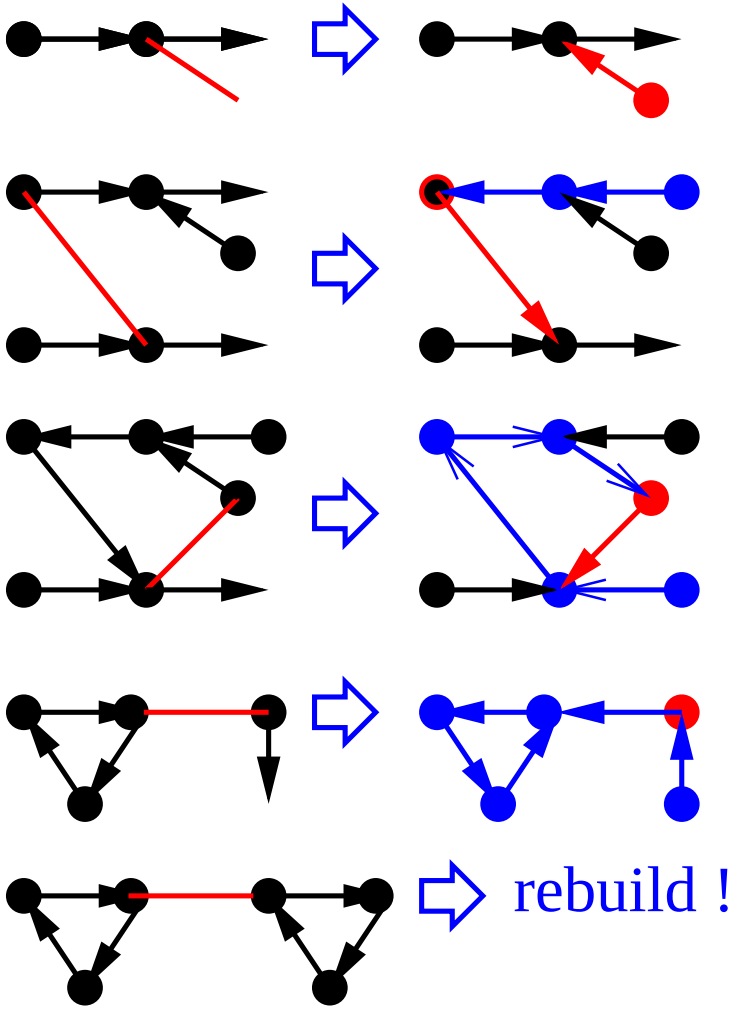
Graph model.

Node: table cells

Undirected edge: element $x \rightsquigarrow$ edge $\{h_1(x), h_2(x)\}$

Directed: $(h_2(x), h_1(x))$ means element x is stored at cell $h_2(x)$

Lemma: insert(x) succeeds iff the component containing $h_1(x), h_2(x)$ contains no more edges than nodes.



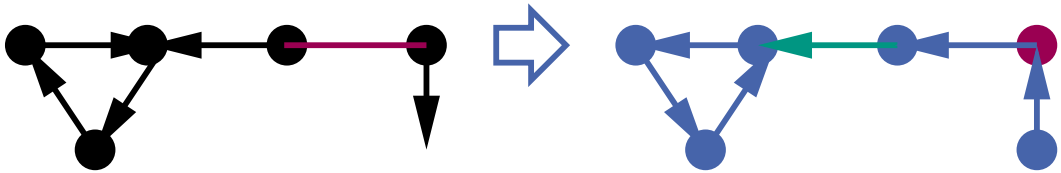
Cuckoo Hashing – Rebuilds

Lemma: insert(x) succeeds iff
the component containing $h_1(x), h_2(x)$
contains no more edges than nodes.

Proof outline: (if-part)

$h_1(x)$ **in tree:** flip path to root

$h_1(x)$ **in pseudotree** p**,** $h_2(x)$ **in tree** t**:**
flip cycle and path to root in t



Cuckoo Hashing – How Many Rebuilds?

Theorem: For truly random hash functions,

$\mathbf{Pr}[\text{rebuild necessary}] = \mathrm{O}(1/n)$

Proof: via random graph theory

Random Graph Theory

[Erdős, Rényi 1959]

$\mathcal{G}(n,m)$:= sample space of all graphs with n nodes, m edges.

A random graph from $\mathcal{G}(n,m)$ has certain properties with high probability, here $\geq 1-\mathrm{O}(1/n)$.

Famous: The evolution of component sizes with increasing m:

$< (1-\varepsilon)n/2$: Trees and pseudotrees of size $\mathrm{O}(\log n)$

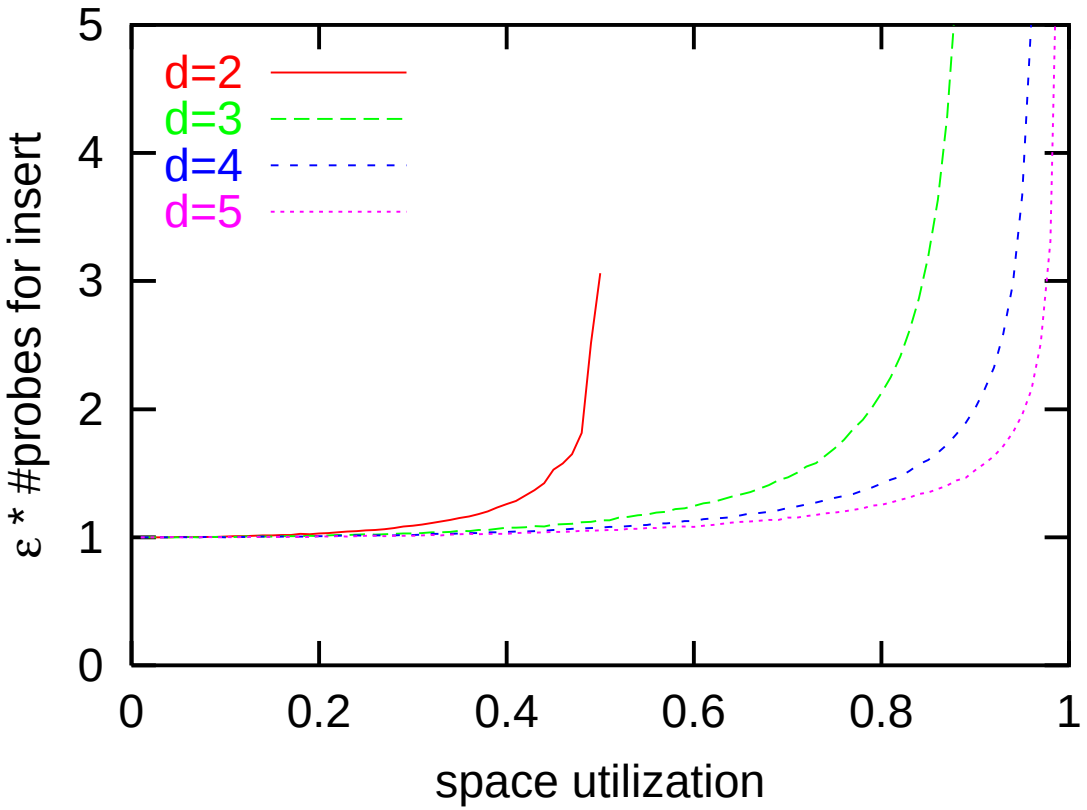
$> (1+\varepsilon)n/2$: A "giant" component of size $\Theta(n)$ (sudden emergence)

$> (1+\varepsilon)n\ln n/2$: One single component

Space Efficient Cuckoo Hashing

***d*-ary:** [Fotakis, Pagh, Sanders, Spirakis 2003] d possible places.

Insertion: BFS, random walk, ...



Space Efficient Cuckoo Hashing

d-ary: [Fotakis, Pagh, Sanders, Spirakis 2003] d possible places.

blocked: [Dietzfelbinger, Weidling 2005] cells house d elements.

Cache efficient !

blocked, d-ary, dynamic growing:

[Maier, Sanders 2017]

Zusammenfassung: Randomisierte Algorithmen

- einfache, effiziente Algorithmen
- oft bestes bekanntes Verfahren
- z.T. Unmöglichkeitsbeweise für deterministische Verfahren
- Analyse oft nichttrivial
- z.T. wird aus esoterischer Theorie ein praxisrelevantes Werkzeug, z.B. random graph evolution
- Las Vegas versus Monte Carlo
- Brücke zur Algebra: z.B. Checker für Sortieren

Ausblick: Randomisierte Algorithmen

- externe minimale Spannbäume
- mehr quicksort (strings, parallel)
- kleinster umschließender Kreis
- online paging