Algorithmen / Algorithms II

Peter Sanders

Exercise:
Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:
http://algo2.iti.kit.edu/AlgorithmenII_WS20.php
6 Randomised Algorithms

Using random (bits) to accelerate/simplify algorithms

Las Vegas: Guarantee a correct result – running time is a random variable already known:

- quicksort
- hashing

Monte Carlo: The result is incorrect with a failure probability $p$

Repeating the algorithm $k$ times decreases the failure-probability exponentially ($p^k$).

Further details in “Randomised Algorithms” by Thomas Worsch
6.1 Sorting – (Result-)Checking

Permutation-Property (sortedness: is trivial)

\[ \langle e_1, \ldots, e_n \rangle \text{ is a permutation of } \langle e'_1, \ldots, e'_n \rangle \text{ exactly when} \]

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

Let \( \mathbb{F} \) be a field, and map : Key \( \rightarrow \mathbb{F} \) is injective.
Observation: \( q \) has at most \( n \) zeros (roots).
Evaluating \( q \) at random position \( x \in \mathbb{F} \).

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

Linear time Monte Carlo algorithm

Question: Which field \( \mathbb{F} \) do we use?
Sort Checking II – with Lorenz Hübschle-Schneider

Is the finite sequence $E$ a permutation of another sequence $E'$?

Let $h$ be a random hash function with destination range $0..U - 1$,

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

**Checker:** return $h(E) = h(E')$
Sort Checking II – with Lorenz Hübschle-Schneider

Is the finite sequence $E$ a permutation of another sequence $E'$?

Let $h$ be a random hash function with destination range $0..U - 1$,

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

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

Correct if $E = E'$.

**Case $E \neq E'$:** We show $\mathbb{P}[h(E) = h(E')] \leq \frac{1}{U}$

Let $e$ be an element, that appears $k \times$ in $E$ and $k' \neq k \times$ in $E'$.

\[
\begin{align*}
    h(E) = h(E') &\iff h(E \setminus e) + kh(e) = h(E' \setminus e) + k'h(e) \\
    &\iff h(e) = \frac{h(E' \setminus e) - h(E \setminus e)}{k - k'} =: x
\end{align*}
\]

$\mathbb{P}[h(e) = x] \leq \frac{1}{U}$ because $x$ is independent of $h(e)$

$\blacksquare$
6.2 Hashing II

Perfect Hashing

Idea: given a set of inputs $S$ make $h$ injective on this set.

This needs $\Omega(n)$ bits of space!
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: $E[T_{\text{find}}] \approx \frac{1}{\varepsilon}$

Dynamic Perfect Hashing,
[Dietzfelbinger et al. 94]
Worst case constant time
for lookup but $\varepsilon$ is not small.

Approaching the Information Theoretic Lower Bound:
[Brodnik Munro 99, Raman Rao 02]
Space $(1 + o(1)) \times$ lower bound without associated information
Simple, fast, $\approx 3$ bits/element [FiRe/FiPHa: Müller, S, Schulze, Zhou 14]
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 \( x \) \( \sim \) 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: \( \text{insert}(x) \) succeeds iff the component containing \( h_1(x), h_2(x) \) contains no more edges than nodes.

\[ \Rightarrow \text{rebuild!} \]
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,

\[ \Pr[\text{rebuild necessary}] = O\left(\frac{1}{n}\right) \]

**Proof:** via random graph theory
Random Graph Theory

\( \mathcal{G}(n, m) := \) 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 \( \geq 1 - O(1/n) \).

Famous: The evolution of component sizes with increasing \( m \):

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

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

\( > (1 + \varepsilon)n\ln n/2 \): One single component
Space Efficient Cuckoo Hashing


\textbf{Insertion}: BFS, random walk, \ldots \\ expected time: $O\left(\frac{1}{\varepsilon}\right)$ ?

![Graph showing space efficiency for different $d$ values](image)
Space Efficient Cuckoo Hashing

\textit{d-ary:} [Fotakis, Pagh, Sanders, Spirakis 2003] \(d\) possible places.

\textit{blocked:} [Dietzfelbinger, Weidling 2005] cells house \(d\) elements.
Cache efficient!

\textit{blocked, \textit{d-ary, dynamic growing:}}

[Maier, Sanders 2017]
Recap – Randomised Algorithms

- Easy, efficient algorithms
- In many cases the best known procedures
- Sometimes deterministic solutions are (provably) impossible
- Often examples for non-trivial analysis
- Sometimes esoteric theory leads to tools that are relevant in practice, e.g. random graph evolution
- Las Vegas versus Monte Carlo
- Bridge to algebra, e.g. Sort-Checker
Lookout – Randomised Algorithms

- External minimum spanning trees
- More quicksort (strings, parallel)
- Smallest enclosing circle
- Online paging