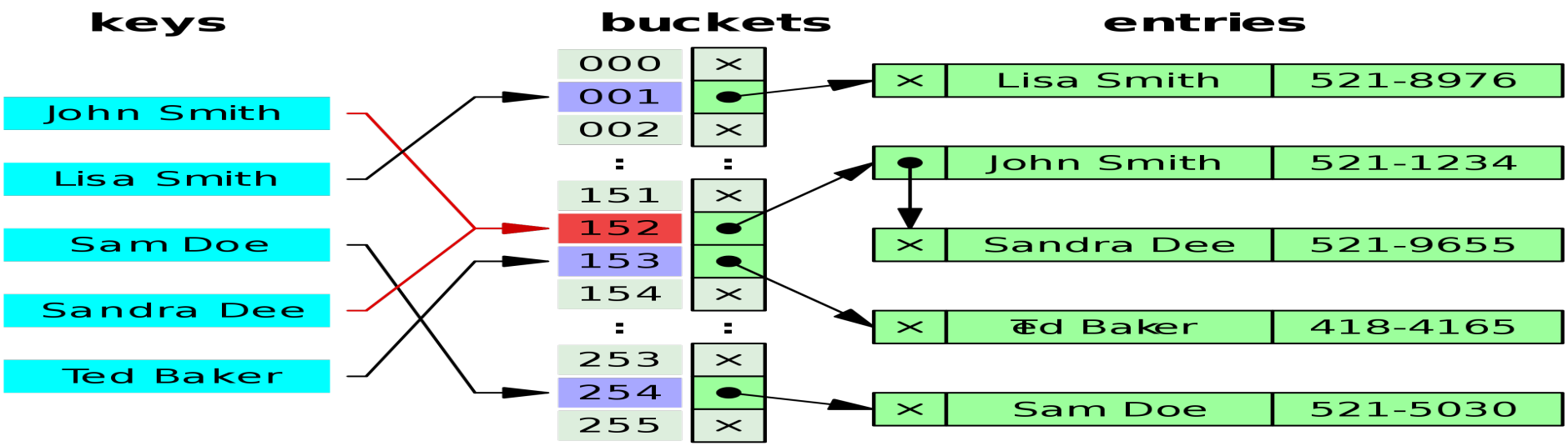
Karlsruher Institut für Technologie

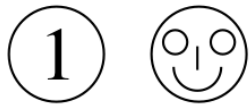
Algorithmen I - Tutorium 4

Sebastian Schmidt – *isibboi@gmail.com*

Arbeitsgruppe Kryptographie und Sicherheit



Wahrscheinlichkeitstheorie



Elementarereignisse Ω

Ereignisse: Teilmengen von Ω

p_x =Wahrscheinlichkeit von $x \in \Omega$. $\sum_x p_x = 1$!

Gleichverteilung: $p_x = \frac{1}{|\Omega|}$

$\mathbb{P}[\mathscr{E}] = \sum_{x \in \mathscr{E}} p_x$

Zufallsvariable (ZV) $X_0 : \Omega \to \mathbb{R}$

0-1-Zufallsvariable (Indikator-ZV) $I : \Omega \to \{0,1\}$

Erwartungswert $\mathrm{E}[X] = \sum_{y \in \Omega} p_y X(y)$

Linearität des Erwartungswerts: $\mathrm{E}[X+Y] = \mathrm{E}[X] + \mathrm{E}[Y]$

Hash-Beispiel

Hash-Funktionen $\{0..m-1\}^{\mathrm{Key}}$

$\mathscr{E}_{42} = \{h \in \Omega : h(4) = h(2)\}$

$p_h = m^{-|\mathrm{Key}|}$

$\mathbb{P}[\mathscr{E}_{42}] = \frac{1}{m}$

$X = |\{e \in M : h(e) = 0\}|$

$\mathrm{E}[X] = \frac{|M|}{m}$

Hashing mit einfach verketteten Listen

- Hashtabelle mit $h_n(x) := x \mod n$
  - Wie groß sollte die Hashtabelle sein?
  - Gegeben $\{36, 78, 50, 1, 92, 15, 43, 99, 64\}$. Füge diese Zahlen in eine Hashtabelle mit Hashfunktion h_5 und h_7 ein.
- Gebt Beispiele für gute und schlechte Hashfunktionen
- Laufzeiten: `insert`, `find` und `remove`

Hashing mit einfach verketteten Listen

- Hashtabelle mit $h_n(x) := x \mod n$
  - Wie groß sollte die Hashtabelle sein?
  - Gegeben $\{36, 78, 50, 1, 92, 15, 43, 99, 64\}$. Füge diese Zahlen in eine Hashtabelle mit Hashfunktion h_5 und h_7 ein.
- Gebt Beispiele für gute und schlechte Hashfunktionen
- Laufzeiten: `insert`, `find` und `remove`

Hashing mit einfach verketteten Listen

- Hashtabelle mit $h_n(x) := x \mod n$
  - Wie groß sollte die Hashtabelle sein?
  - Gegeben $\{36, 78, 50, 1, 92, 15, 43, 99, 64\}$. Füge diese Zahlen in eine Hashtabelle mit Hashfunktion h_5 und h_7 ein.
- Gebt Beispiele für gute und schlechte Hashfunktionen
- Laufzeiten: `insert`, `find` und `remove`

Hashing mit einfach verketteten Listen

- Hashtabelle mit $h_n(x) := x \mod n$
  - Wie groß sollte die Hashtabelle sein?
  - Gegeben $\{36, 78, 50, 1, 92, 15, 43, 99, 64\}$. Füge diese Zahlen in eine Hashtabelle mit Hashfunktion h_5 und h_7 ein.
- Gebt Beispiele für gute und schlechte Hashfunktionen
- Laufzeiten: `insert`, `find` und `remove`

Hashing mit einfach verketteten Listen

- Hashtabelle mit $h_n(x) := x \mod n$
  - Wie groß sollte die Hashtabelle sein?
  - Gegeben $\{36, 78, 50, 1, 92, 15, 43, 99, 64\}$. Füge diese Zahlen in eine Hashtabelle mit Hashfunktion h_5 und h_7 ein.
- Gebt Beispiele für gute und schlechte Hashfunktionen
- Laufzeiten: `insert`, `find` und `remove`

Anwendungsbeispiele für Hashtabellen

Hashtabellen sind besser als Bäume, wenn man die erwartete Laufzeit betrachtet.

- Fallen euch konkrete Beispiele oder Gegenbeispiele ein?

Anwendungsbeispiele für Hashtabellen

Hashtabellen sind besser als Bäume, wenn man die erwartete Laufzeit betrachtet.

- Fallen euch konkrete Beispiele oder Gegenbeispiele ein?

Beweis über Worst-Case

Seien $m, n \in \mathbb{N}$. n ist die Anzahl der Elemente, die in eine Hashtabelle der Größe m eingefügt werden.
Sei U ein Universum mit $|U| \geq mn$.
Zeige, dass eine Teilmenge $M \subset U$ existiert mit $|M| \geq n$, sodass alle Elemente aus M dem selben Slot der Hashtabelle zugeordnet werden.

Verbesserung des Worst-Case

„Nach dem dritten Glas Bier behauptet ein Kommilitone, man könne Hashing mit verketteten Listen entscheidend verbessern, indem man die verketteten Listen stets sortiert halte.“

- Wie ändert sich dadurch die Worst-Case Laufzeit von `insert`, `find` und `remove`?
- Was muss man tun, um die Sortiertheit auszunutzen?
- Wie ändert sich nun die Laufzeit?
- Kann man durch Amortisierung noch was verbessern?

Verbesserung des Worst-Case

„Nach dem dritten Glas Bier behauptet ein Kommilitone, man könne Hashing mit verketteten Listen entscheidend verbessern, indem man die verketteten Listen stets sortiert halte.“

- Wie ändert sich dadurch die Worst-Case Laufzeit von `insert`, `find` und `remove`?
- Was muss man tun, um die Sortiertheit auszunutzen?
- Wie ändert sich nun die Laufzeit?
- Kann man durch Amortisierung noch was verbessern?

Verbesserung des Worst-Case

„Nach dem dritten Glas Bier behauptet ein Kommilitone, man könne Hashing mit verketteten Listen entscheidend verbessern, indem man die verketteten Listen stets sortiert halte.“

- Wie ändert sich dadurch die Worst-Case Laufzeit von `insert`, `find` und `remove`?
- Was muss man tun, um die Sortiertheit auszunutzen?
- Wie ändert sich nun die Laufzeit?
- Kann man durch Amortisierung noch was verbessern?

Verbesserung des Worst-Case

„Nach dem dritten Glas Bier behauptet ein Kommilitone, man könne Hashing mit verketteten Listen entscheidend verbessern, indem man die verketteten Listen stets sortiert halte.“

- Wie ändert sich dadurch die Worst-Case Laufzeit von `insert`, `find` und `remove`?
- Was muss man tun, um die Sortiertheit auszunutzen?
- Wie ändert sich nun die Laufzeit?
- Kann man durch Amortisierung noch was verbessern?

Sparse Array

Ein Sparse Array ist eine Datenstruktur mit den Eigenschaften eines beschränkten Arrays und zusätzlich:

- Die Erzeugung eines leeren Sparse Arrays mit n Slots braucht $O(1)$ Zeit
- Es gibt eine Operation `reset`, die das Array in $O(1)$ Zeit leert
- Das Sparse Array unterstützt `get(i)` und `set(i, x)` in $O(1)$ Zeit
  - `get(i)`: Gibt das Element im Slot i zurück, oder $\perp$, wenn der Slot leer ist
  - `set(i, x)`: Speichert x im Slot i

Wir können beliebig viel uninitialisierten Speicher in $O(1)$ Zeit allokieren.
Ein Sparse Array mit n Slots braucht $O(n)$ Speicher.

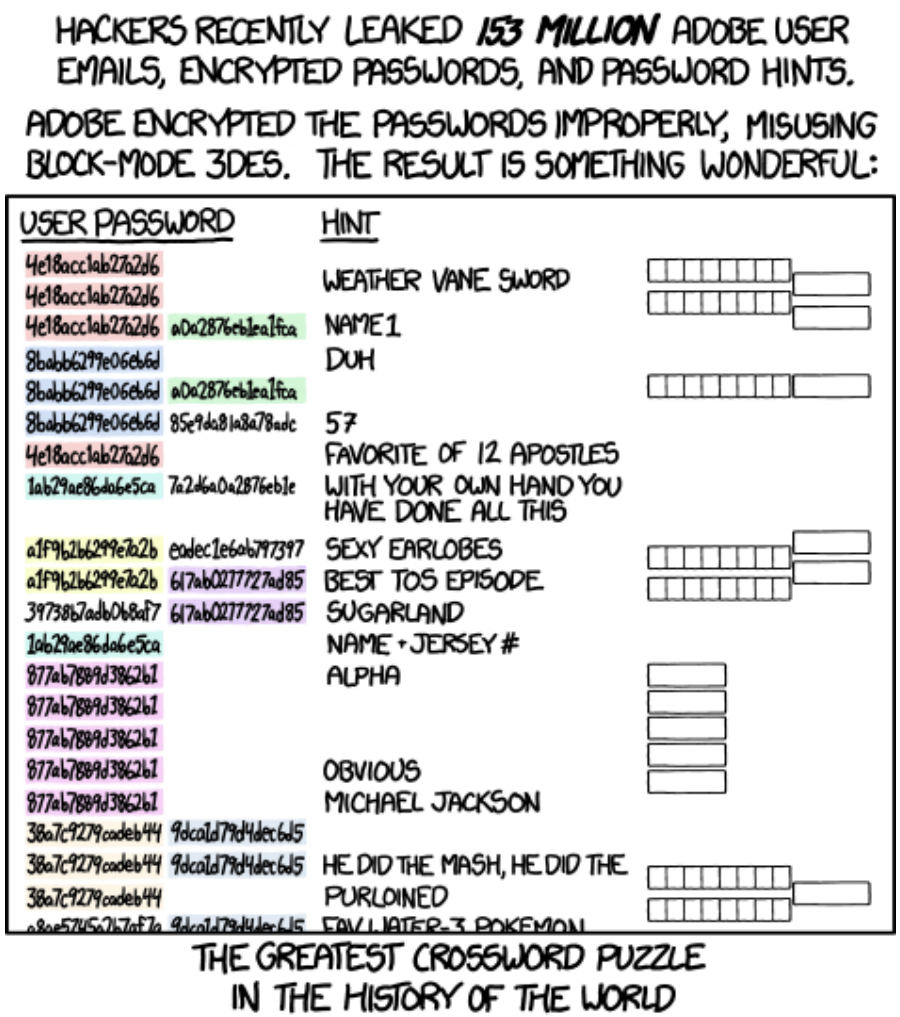
Sparse Array

- Geht unsere Realisierung für große Datentypen sparsam mit dem Speicher um? Wenn nein, kann man sie noch verbessern?
- Welche Vor- und Nachteile im Bezug auf das Iterieren und den Speicherverbrauch hat das Sparse Array gegenüber beschränkten Arrays?

Sparse Array

- Geht unsere Realisierung für große Datentypen sparsam mit dem Speicher um? Wenn nein, kann man sie noch verbessern?
- Welche Vor- und Nachteile im Bezug auf das Iterieren und den Speicherverbrauch hat das Sparse Array gegenüber beschränkten Arrays?

Ende!


HACKERS RECENTLY LEAKED 153 MILLION ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS.
ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:
USER PASSWORD
HINT
4e18acc1ab27a2d6
WEATHER VANE SWORD
4e18acc1ab27a2d6
4e18acc1ab27a2d6 a0a2876eb1ea1fca
NAME1
8babb6299e06eb6d
DUH
8babb6299e06eb6d a0a2876eb1ea1fca
8babb6299e06eb6d 85c9da81a8a78adc
57
4e18acc1ab27a2d6
FAVORITE OF 12 APOSTLES
1ab29ae86da6e5ca 7a2d6a0a2876eb1e
WITH YOUR OWN HAND YOU HAVE DONE ALL THIS
a1f9b2b6299e7a2b eadec1e6ab797397
SEXY EARLOBES
a1f9b2b6299e7a2b 617ab0277727ad85
BEST TOS EPISODE
3973867adb068af7 617ab0277727ad85
SUGARLAND
1ab29ae86da6e5ca
NAME + JERSEY #
877ab7889d3862b1
ALPHA
877ab7889d3862b1
877ab7889d3862b1
877ab7889d3862b1
OBVIOUS
877ab7889d3862b1
MICHAEL JACKSON
38a7c9279cadeb44 9dca1d79d4dec6d5
38a7c9279cadeb44 9dca1d79d4dec6d5
HE DID THE MASH, HE DID THE
38a7c9279cadeb44
PURLOINED
THE GREATEST CROSSWORD PUZZLE IN THE HISTORY OF THE WORLD