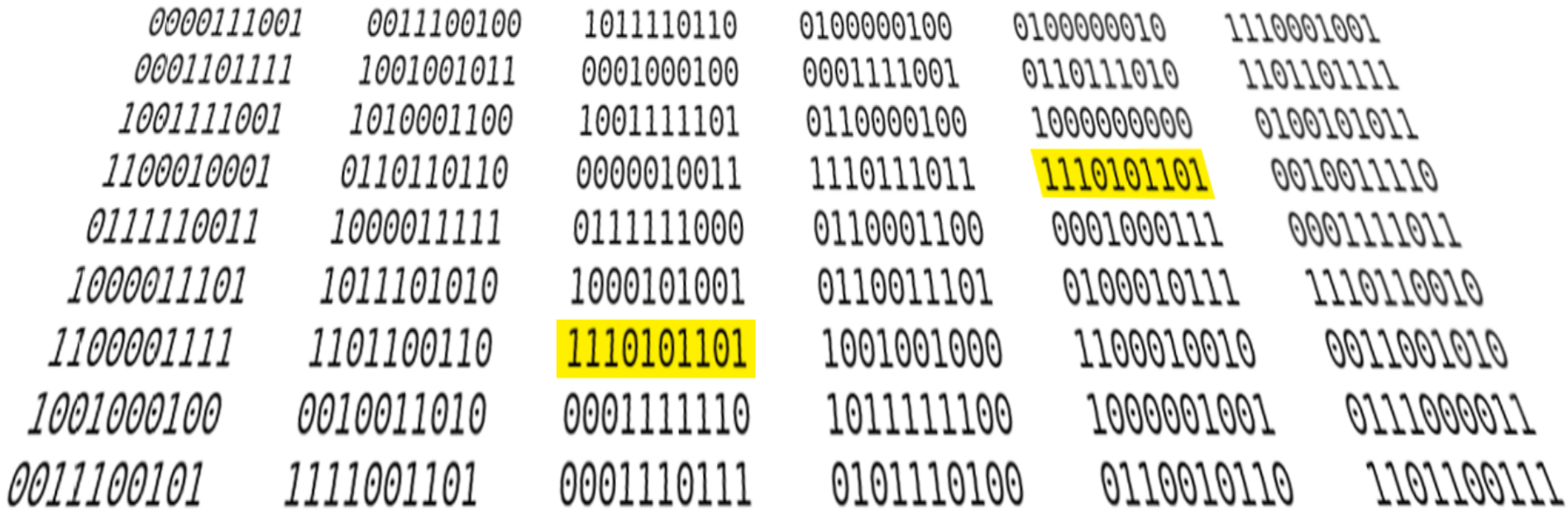
Karlsruher Institut für Technologie

Tutorium 4: Hashing

Holger Ebhart | 13. Mai 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15



Gliederung

3. Übungsblatt

Grundlagen

Ziehen von k Kugeln aus einer Urne mit n Kugeln

- Kugeln unterscheidbar (Beachtung der Reihenfolge)
  - mit Zurücklegen: n^k
  - ohne Zurücklegen: $\frac{n!}{(n-k)!} = n \cdot (n-1) \cdot \ldots \cdot (n-(k-1))$
- Kugeln nicht unterscheidbar (Reihenfolge egal)
  - mit Zurücklegen: $\binom{n+k-1}{k}$
  - ohne Zurücklegen: $\binom{n}{k}$

Aufgaben

Berechne Folgendes:

- Wie viele verschiedene Wörter w lassen sich aus $\{A, B, M, N, S\}$ mit $|w| = 3$ bilden?
- Wie viele, wenn kein Buchstabe doppelt vorkommen darf?
- Wie viele neue Wörter können aus “ANANAS“ gebildet werden?

Aufgaben

Ein Parkplatz habe 18 Plätze, welche folgendermaßen belegt sind: 6 Opel, 2 Fiat, 4 Volvo, 5 Skoda und 1 Smart. Die Fahrzeuge sind durch ihre Nummernschilder unterscheidbar.

Wie viele Möglichkeiten gibt es wenn

- alle Skodas nebeneinander stehen?
- alle Fahrzeuge der gleichen Marke nebeneinander stehen?

Aufgaben

Ein Parkplatz habe 18 Plätze, welche folgendermaßen belegt sind: 6 Opel, 2 Fiat, 4 Volvo, 5 Skoda und 1 Smart. Die Fahrzeuge sind durch ihre Nummernschilder unterscheidbar.

Wie viele Möglichkeiten gibt es wenn

- alle Skodas nebeneinander stehen?
- alle Fahrzeuge der gleichen Marke nebeneinander stehen?

Aufgaben

Ein Parkplatz habe 18 Plätze, welche folgendermaßen belegt sind: 6 Opel, 2 Fiat, 4 Volvo, 5 Skoda und 1 Smart. Die Fahrzeuge sind durch ihre Nummernschilder unterscheidbar.

Wie viele Möglichkeiten gibt es wenn

- alle Skodas nebeneinander stehen?
- alle Fahrzeuge der gleichen Marke nebeneinander stehen?

Universelles Hashing

- $\mathcal{H} \subseteq \{0 \dots m-1\}^{Key}$
- $h \in \mathcal{H}$ zufällig
- $\forall x, y \in Key : x \neq y$
- $\mathbb{P}[h(x) = h(y)] = \frac{1}{m}$

Ist die folgende Hashfunktion h universell?

- $h : \{2^n | n \in \mathbb{N}\} \to \{0, \dots, 9\}$
  $x \to x \mod 10$

Universelles Hashing

- $\mathcal{H} \subseteq \{0 \dots m-1\}^{Key}$
- $h \in \mathcal{H}$ zufällig
- $\forall x, y \in Key : x \neq y$
- $\mathbb{P}[h(x) = h(y)] = \frac{1}{m}$

Ist die folgende Hashfunktion h universell?

- $h : \{2^n | n \in \mathbb{N}\} \to \{0, \dots, 9\}$
  $x \to x \mod 10$

Hashing mit linearer Suche

- verwende Array mit mind. n + 1 Slots
- insert(e) $\rightarrow$ $h(key(e))$ frei?
  - ja: Einfügen von e
  - nein: suche nächsten freien Platz und füge e dort ein
- find(k)
  Suche ab $h(k)$ nach einem Element x mit $key(x) = k$. Ende spätestens bei einer Lücke.
- remove(k) $\rightarrow$ find(k) + löschen

Hashing mit linearer Suche

- verwende Array mit mind. n + 1 Slots
- insert(e) $\rightarrow$ $h(key(e))$ frei?

  ja: Einfügen von e

  nein: suche nächsten freien Platz und füge e dort ein
- find(k)

  Suche ab $h(k)$ nach einem Element x mit $key(x) = k$. Ende spätestens bei einer Lücke.
- remove(k) $\rightarrow$ find(k) + löschen

Hashing mit linearer Suche

- verwende Array mit mind. n + 1 Slots
- insert(e) $\rightarrow$ $h(key(e))$ frei?

  ja: Einfügen von e

  nein: suche nächsten freien Platz und füge e dort ein
- find(k)

  Suche ab $h(k)$ nach einem Element x mit $key(x) = k$. Ende spätestens bei einer Lücke.
- remove(k) $\rightarrow$ find(k) + löschen

Hashing mit linearer Suche

- verwende Array mit mind. n + 1 Slots
- insert(e) $\rightarrow$ $h(key(e))$ frei?

  ja: Einfügen von e

  nein: suche nächsten freien Platz und füge e dort ein
- find(k)

  Suche ab $h(k)$ nach einem Element x mit $key(x) = k$. Ende spätestens bei einer Lücke.
- remove(k) $\rightarrow$ find(k) + löschen

Beispiel

- 4 Slots
- $h(k) = (k + 1) \mod 5$
- $key(e) = e$
- Führe folgende Operationen aus:
  1. Einfügen von 7, 8, 13, 2
  2. Suche nach 13, 2
  3. Entfernen von 13, 7

Beispiel

- 4 Slots
- $h(k) = (k + 1) \mod 5$
- $key(e) = e$
- Führe folgende Operationen aus:
  1. Einfügen von 7, 8, 13, 2
  2. Suche nach 13, 2
  3. Entfernen von 13, 7

Beispiel

- 4 Slots
- $h(k) = (k + 1) \mod 5$
- $key(e) = e$
- Führe folgende Operationen aus:
  1. Einfügen von 7, 8, 13, 2
  2. Suche nach 13, 2
  3. Entfernen von 13, 7

Vergleich

Operation	Array	verkettete Listen	lineare Suche
create	$\mathrm{O}(1)$ / $\mathrm{O}(n)$	$\mathrm{O}(1)$ / $\mathrm{O}(n)$	$\mathrm{O}(1)$ / $\mathrm{O}(n)$
insert	$\mathrm{O}(1)$ (*)	$\mathrm{O}(1)$	amortisiert $\mathrm{O}(1)$
find	$\mathrm{O}(1)$ (*)	erwartet $\mathrm{O}(1)$	(amortisiert) $\mathrm{O}(1)$
remove	$\mathrm{O}(1)$ (*)	erwartet $\mathrm{O}(1)$	(amortisiert) $\mathrm{O}(1)$

*: Es existiert nur ein Slot pro Wert den die Hashfunktion annehmen kann

$\rightarrow$ Kollision

Aufgabe

a) Entwerfen Sie ein *SparseArray* („spärlich besetztes Array“). Dabei handelt es sich um eine Datenstruktur mit den Eigenschaften eines beschränkten Arrays, die zusätzlich schnelle Erzeugung und schnellen Reset ermöglicht. Nehmen Sie dabei an, dass **allocate** beliebig viel *uninitialisierten* Speicher in konstanter Zeit liefert. Das *SparseArray* habe folgende Eigenschaften:
   - Ein *SparseArray* mit n Slots braucht $O(n)$ Speicher.
   - Erzeugen eines leeren *SparseArray* mit n Slots braucht $O(1)$ Zeit.
   - Das *SparseArray* unterstützt eine Operation *reset*, die es in $O(1)$ Zeit in leeren Zustand versetzt.
   - Das *SparseArray* unterstützt die Operation $get(i)$ und $set(i, x)$. Dabei liefert $A.get(i)$ den Wert, der sich im i-ten Slot des *SparseArray* A befindet; $A.set(i, x)$ setzt das Element im i-ten Slot auf den Wert x. Wurde der i-te Slot seit der Erzeugung bzw. dem letzten *reset* noch nicht mit auf einen bestimmten Wert gesetzt, so liefert $A.get(i)$ einen speziellen Wert $\perp$. Die Operationen *get* und *set* dürfen zudem beide nicht mehr als $O(1)$ Zeit verbrauchen (*random access*).

Aufgabe

a) Entwerfen Sie ein *SparseArray* („spärlich besetztes Array“). Dabei handelt es sich um eine Datenstruktur mit den Eigenschaften eines beschränkten Arrays, die zusätzlich schnelle Erzeugung und schnellen Reset ermöglicht. Nehmen Sie dabei an, dass **allocate** beliebig viel *uninitialisierten* Speicher in konstanter Zeit liefert. Das *SparseArray* habe folgende Eigenschaften:

- Ein *SparseArray* mit n Slots braucht $\mathrm{O}(n)$ Speicher.
- Erzeugen eines leeren *SparseArray* mit n Slots braucht $\mathrm{O}(1)$ Zeit.
- Das *SparseArray* unterstützt eine Operation *reset*, die es in $\mathrm{O}(1)$ Zeit in leeren Zustand versetzt.
- Das *SparseArray* unterstützt die Operation $get(i)$ und $set(i, x)$. Dabei liefert $A.get(i)$ den Wert, der sich im i-ten Slot des *SparseArray* A befindet; $A.set(i, x)$ setzt das Element im i-ten Slot auf den Wert x. Wurde der i-te Slot seit der Erzeugung bzw. dem letzten *reset* noch nicht mit auf einen bestimmten Wert gesetzt, so liefert $A.get(i)$ einen speziellen Wert $\perp$. Die Operationen *get* und *set* dürfen zudem beide nicht mehr als $\mathrm{O}(1)$ Zeit verbrauchen (*random access*).

Aufgabe

a) Entwerfen Sie ein *SparseArray* („spärlich besetztes Array“). Dabei handelt es sich um eine Datenstruktur mit den Eigenschaften eines beschränkten Arrays, die zusätzlich schnelle Erzeugung und schnellen Reset ermöglicht. Nehmen Sie dabei an, dass **allocate** beliebig viel *uninitialisierten* Speicher in konstanter Zeit liefert. Das *SparseArray* habe folgende Eigenschaften:
   - Ein *SparseArray* mit n Slots braucht $\mathrm{O}(n)$ Speicher.
   - Erzeugen eines leeren *SparseArray* mit n Slots braucht $\mathrm{O}(1)$ Zeit.
   - Das *SparseArray* unterstützt eine Operation *reset*, die es in $\mathrm{O}(1)$ Zeit in leeren Zustand versetzt.
   - Das *SparseArray* unterstützt die Operation $get(i)$ und $set(i, x)$. Dabei liefert $A.get(i)$ den Wert, der sich im i-ten Slot des *SparseArray* A befindet; $A.set(i, x)$ setzt das Element im i-ten Slot auf den Wert x. Wurde der i-te Slot seit der Erzeugung bzw. dem letzten *reset* noch nicht mit auf einen bestimmten Wert gesetzt, so liefert $A.get(i)$ einen speziellen Wert $\perp$. Die Operationen *get* und *set* dürfen zudem beide nicht mehr als $\mathrm{O}(1)$ Zeit verbrauchen (*random access*).

Aufgabe

a) Entwerfen Sie ein *SparseArray* („spärlich besetztes Array“). Dabei handelt es sich um eine Datenstruktur mit den Eigenschaften eines beschränkten Arrays, die zusätzlich schnelle Erzeugung und schnellen Reset ermöglicht. Nehmen Sie dabei an, dass **allocate** beliebig viel *uninitialisierten* Speicher in konstanter Zeit liefert. Das *SparseArray* habe folgende Eigenschaften:
   - Ein *SparseArray* mit n Slots braucht $\mathrm{O}(n)$ Speicher.
   - Erzeugen eines leeren *SparseArray* mit n Slots braucht $\mathrm{O}(1)$ Zeit.
   - Das *SparseArray* unterstützt eine Operation *reset*, die es in $\mathrm{O}(1)$ Zeit in leeren Zustand versetzt.
   - Das *SparseArray* unterstützt die Operation $get(i)$ und $set(i, x)$. Dabei liefert $A.get(i)$ den Wert, der sich im i-ten Slot des *SparseArray* A befindet; $A.set(i, x)$ setzt das Element im i-ten Slot auf den Wert x. Wurde der i-te Slot seit der Erzeugung bzw. dem letzten *reset* noch nicht mit auf einen bestimmten Wert gesetzt, so liefert $A.get(i)$ einen speziellen Wert $\perp$. Die Operationen *get* und *set* dürfen zudem beide nicht mehr als $\mathrm{O}(1)$ Zeit verbrauchen (*random access*).

Aufgabe

a) Entwerfen Sie ein *SparseArray* („spärlich besetztes Array“). Dabei handelt es sich um eine Datenstruktur mit den Eigenschaften eines beschränkten Arrays, die zusätzlich schnelle Erzeugung und schnellen Reset ermöglicht. Nehmen Sie dabei an, dass **allocate** beliebig viel *uninitialisierten* Speicher in konstanter Zeit liefert. Das *SparseArray* habe folgende Eigenschaften:
   - Ein *SparseArray* mit n Slots braucht $\mathrm{O}(n)$ Speicher.
   - Erzeugen eines leeren *SparseArray* mit n Slots braucht $\mathrm{O}(1)$ Zeit.
   - Das *SparseArray* unterstützt eine Operation *reset*, die es in $\mathrm{O}(1)$ Zeit in leeren Zustand versetzt.
   - Das *SparseArray* unterstützt die Operation *get*(i) und *set*(i, x). Dabei liefert A.*get*(i) den Wert, der sich im i-ten Slot des *SparseArray* A befindet; A.*set*(i, x) setzt das Element im i-ten Slot auf den Wert x. Wurde der i-te Slot seit der Erzeugung bzw. dem letzten *reset* noch nicht mit auf einen bestimmten Wert gesetzt, so liefert A.*get*(i) einen speziellen Wert $\perp$. Die Operationen *get* und *set* dürfen zudem beide nicht mehr als $\mathrm{O}(1)$ Zeit verbrauchen (*random access*).

Aufgabe

b) Nehmen Sie an, dass die Datenelemente, die Sie im *SparseArray* ablegen wollen, recht groß sind (also z.B. Records mit 10, 20 oder mehr Einträgen). Geht Ihre Realisierung unter dieser Annahme sparsam oder verschwenderisch mit dem Speicherplatz um? Wenn Sie Ihre Realisierung für verschwenderisch halten, überlegen Sie ob und wie Sie es besser machen können.

c) Vergleichen Sie Ihr *SparseArray* mit Bounded-Arrays. Welche Vorteile und Nachteile sehen Sie im Hinblick auf den Speicherverbrauch und auf das Iterieren über alle Elemente mit *set* eingefügten Elemente?

Aufgabe

b) Nehmen Sie an, dass die Datenelemente, die Sie im *SparseArray* ablegen wollen, recht groß sind (also z.B. Records mit 10, 20 oder mehr Einträgen). Geht Ihre Realisierung unter dieser Annahme sparsam oder verschwenderisch mit dem Speicherplatz um? Wenn Sie Ihre Realisierung für verschwenderisch halten, überlegen Sie ob und wie Sie es besser machen können.

c) Vergleichen Sie Ihr *SparseArray* mit Bounded-Arrays. Welche Vorteile und Nachteile sehen Sie im Hinblick auf den Speicherverbrauch und auf das Iterieren über alle Elemente mit *set* eingefügten Elemente?

Hashtabelle

- Es sei die Hashfunktion $h(key) = (key * 3) \mod 9$ gegeben
- Es sei: $key : Element \rightarrow \{0, 1, \ldots, 9\}$
- Füge die Elemente 1,5,9,23,44,65,78,91 in eine Hashtabelle
- Benutze Hashing mit verketteten Listen
- Benutze Hashing mit linearer Suche

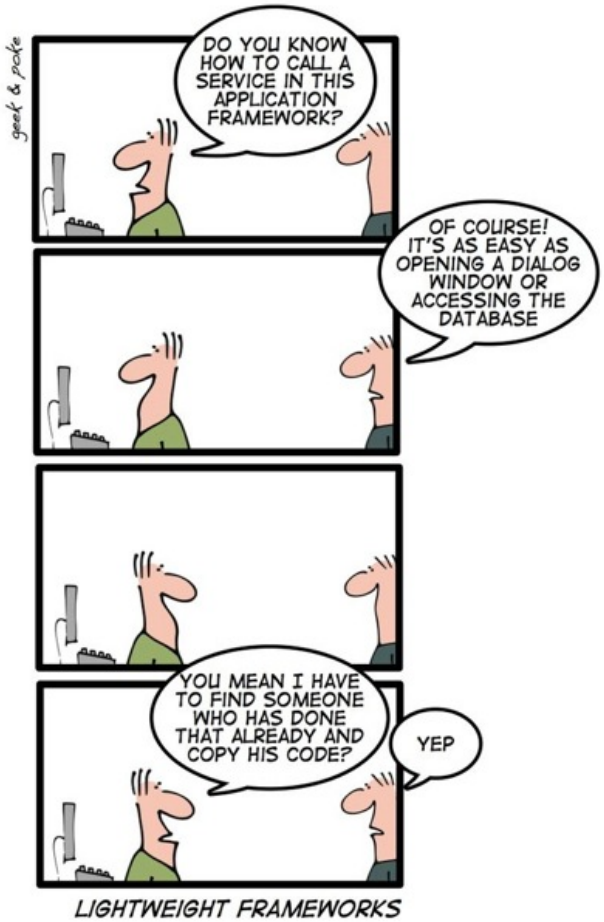
Hashtabelle

- Es sei die Hashfunktion $h(key) = (key * 3) \mod 9$ gegeben
- Es sei: $key : Element \rightarrow \{0, 1, \ldots, 9\}$
- Füge die Elemente 1,5,9,23,44,65,78,91 in eine Hashtabelle
- Benutze Hashing mit verketteten Listen
- Benutze Hashing mit linearer Suche

Hashtabelle

- Es sei die Hashfunktion $h(key) = (key * 3) \mod 9$ gegeben
- Es sei: $key : Element \rightarrow \{0, 1, \ldots, 9\}$
- Füge die Elemente 1,5,9,23,44,65,78,91 in eine Hashtabelle
- Benutze Hashing mit verketteten Listen
- Benutze Hashing mit linearer Suche

Vielen Dank für eure Aufmerksamkeit!
Bis zum nächsten Mal.



datamation.com