

Hashverfahren

Reiner Hüchting

18. April 2023

Themenüberblick

Hashmaps

Hashfunktionen

Themenüberblick

Hashmaps

Hashfunktionen

Hashmaps

Erinnerung: Binäre Suchbäume

- ▶ effiziente Speicherung von Schlüssel-Wert-Paaren
- ▶ schnelles Einfügen, Löschen und Suchen
(z.B. $O(\log n)$ bei AVL-Bäumen)
- ▶ Pointerstrukturen mit bekannten Vor- und Nachteilen

Hashmaps

Erinnerung: Binäre Suchbäume

- ▶ effiziente Speicherung von Schlüssel-Wert-Paaren
- ▶ schnelles Einfügen, Löschen und Suchen
(z.B. $O(\log n)$ bei AVL-Bäumen)
- ▶ Pointerstrukturen mit bekannten Vor- und Nachteilen

Erinnerung: Heaps

- ▶ vollständige Binärbäume mit Teil-Sortierung der Elemente
- ▶ Einfügen und Löschen in $O(\log n)$
- ▶ Zugriff auf Wurzel sogar in $O(1)$
- ▶ Speicherung als Array, effizienter als binäre Suchbäume

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Wie könnte das gehen?

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Wie könnte das gehen?

- ▶ Warum geht der Zugriff auf die Wurzel bei Heaps schnell?

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Wie könnte das gehen?

- ▶ Warum geht der Zugriff auf die Wurzel bei Heaps schnell?
 - ▶ Position ist bekannt und muss nicht gesucht werden!

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Wie könnte das gehen?

- ▶ Warum geht der Zugriff auf die Wurzel bei Heaps schnell?
 - ▶ Position ist bekannt und muss nicht gesucht werden!
- ▶ Können wir das für alle Elemente erreichen?

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Wie könnte das gehen?

- ▶ Warum geht der Zugriff auf die Wurzel bei Heaps schnell?
 - ▶ Position ist bekannt und muss nicht gesucht werden!
- ▶ Können wir das für alle Elemente erreichen?
 - ▶ Idee: Verwende ein Array und berechne die Position des Elements aus dem Element selbst.

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Wie könnte das gehen?

- ▶ Warum geht der Zugriff auf die Wurzel bei Heaps schnell?
 - ▶ Position ist bekannt und muss nicht gesucht werden!
- ▶ Können wir das für alle Elemente erreichen?
 - ▶ Idee: Verwende ein Array und berechne die Position des Elements aus dem Element selbst.

Welchen Preis müssen wir dafür bezahlen?

Hashmaps

Neues Ziel: Average-Case-Zugriff auf jeden Schlüssel in $O(1)$

Wie könnte das gehen?

- ▶ Warum geht der Zugriff auf die Wurzel bei Heaps schnell?
 - ▶ Position ist bekannt und muss nicht gesucht werden!
- ▶ Können wir das für alle Elemente erreichen?
 - ▶ Idee: Verwende ein Array und berechne die Position des Elements aus dem Element selbst.

Welchen Preis müssen wir dafür bezahlen?

- ▶ Erheblich mehr Speicherverbrauch
- ▶ schlechtere Worst-Case-Komplexität

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einem Array (**Hashtabelle**).
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einem Array (**Hashtabelle**).
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Eigenschaften der Hashfunktion

- ▶ Eingabe: Das Element bzw. dessen Schlüssel
- ▶ Ergebnis: Die Position des Elements im Array oder eine Zahl, aus der diese Position berechnet werden kann.

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beispiele für einfache Hashfunktionen

Eingabe	Ergebnis
ein String	Summe der ASCII-Werte

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beispiele für einfache Hashfunktionen

Eingabe	Ergebnis
ein String	Summe der ASCII-Werte
ein String	nach Position gewichtete Summe der ASCII-Werte

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beispiele für einfache Hashfunktionen

Eingabe	Ergebnis
ein String	Summe der ASCII-Werte
ein String	nach Position gewichtete Summe der ASCII-Werte
ein Integer	der Wert selbst

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beispiele für einfache Hashfunktionen

Eingabe	Ergebnis
ein String	Summe der ASCII-Werte
ein String	nach Position gewichtete Summe der ASCII-Werte
ein Integer	der Wert selbst
ein Integer	(gewichtete) Quersumme

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beispiele für einfache Hashfunktionen

Eingabe	Ergebnis
ein String	Summe der ASCII-Werte
ein String	nach Position gewichtete Summe der ASCII-Werte
ein Integer	der Wert selbst
ein Integer	(gewichtete) Quersumme
ein Struct	Summe der Hashwerte aller Member

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer Hashfunktion.

Beobachtungen

- ▶ Hashfunktion berechnet Zahlen aus beliebigen Elementen.

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beobachtungen

- ▶ Hashfunktion berechnet Zahlen aus beliebigen Elementen.
- ▶ Ziel-Position kann z.B. durch *Modulo* errechnet werden.

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beobachtungen

- ▶ Hashfunktion berechnet Zahlen aus beliebigen Elementen.
- ▶ Ziel-Position kann z.B. durch *Modulo* errechnet werden.
- ▶ Die Hashfunktion ist i.d.R. **nicht injektiv**.
- ▶ D.h. es sind **Kollisionen** möglich.

Hashmaps

Idee zu Hashmaps

- ▶ Speichere Elemente in einer Hashtabelle.
- ▶ Berechne Position des Schlüssels mittels einer **Hashfunktion**.

Beobachtungen

- ▶ Hashfunktion berechnet Zahlen aus beliebigen Elementen.
- ▶ Ziel-Position kann z.B. durch *Modulo* errechnet werden.
- ▶ Die Hashfunktion ist i.d.R. **nicht injektiv**.
- ▶ D.h. es sind **Kollisionen** möglich.
- ▶ Anforderung: **Berechnung muss schnell gehen!**

Themenüberblick

Hashmaps

Hashfunktionen

Hashfunktionen

Anforderungen an Hashfunktionen

- ▶ Hashfunktion berechnet Zahlen aus beliebigen Elementen.
- ▶ Kollisionen sind nicht zu vermeiden, sollten aber so selten wie möglich auftreten.
- ▶ Berechnung des Hashes muss schnell gehen.

Hashfunktionen

Anforderungen an Hashfunktionen

- ▶ Hashfunktion berechnet Zahlen aus beliebigen Elementen.
- ▶ Kollisionen sind nicht zu vermeiden, sollten aber so selten wie möglich auftreten.
- ▶ Berechnung des Hashes muss schnell gehen.

Vermeidung von Kollisionen

- ▶ Werte sollten möglichst gleichmäßig verteilt sein.
- ▶ Viel mehr Speicher verwenden als benötigt wird, damit die Wahrscheinlichkeit einer Kollision gering ist.

Hashfunktionen

Anforderungen an Hashfunktionen

- ▶ Hashfunktion berechnet Zahlen aus beliebigen Elementen.
- ▶ Kollisionen sind nicht zu vermeiden, sollten aber so selten wie möglich auftreten.
- ▶ Berechnung des Hashes muss schnell gehen.

Vermeidung von Kollisionen

- ▶ Werte sollten möglichst gleichmäßig verteilt sein.
- ▶ Viel mehr Speicher verwenden als benötigt wird, damit die Wahrscheinlichkeit einer Kollision gering ist.

Umgang mit Kollisionen

- ▶ **geschlossenes Hashing**: Neue Position berechnen (*Sondieren*)
- ▶ **offenes Hashing** Mehrere Elemente pro Position erlauben
 - ▶ Z.B. als Liste pro Position

Hashfunktionen

geschlossenes Hashing

- ▶ Berechne so lange neue Hash-Werte, bis eine freie Stelle in der Hashtabelle gefunden wurde.
- ▶ Auch beim Suchen nach Schlüsseln müssen wiederholt Hashes berechnet und die gefundenen Elemente geprüft werden.

Hashfunktionen

geschlossenes Hashing

- ▶ Berechne so lange neue Hash-Werte, bis eine freie Stelle in der Hashtabelle gefunden wurde.
- ▶ Auch beim Suchen nach Schlüsseln müssen wiederholt Hashes berechnet und die gefundenen Elemente geprüft werden.

Beispiel: Doppel-Hashing

- ▶ Weiche bei Kollisionen auf eine zweite Hashfunktion aus.
- ▶ Multipliziere die Hash-Werte mit der Anzahl der Versuche.

Hashfunktionen

geschlossenes Hashing

- ▶ Berechne so lange neue Hash-Werte, bis eine freie Stelle in der Hashtabelle gefunden wurde.
- ▶ Auch beim Suchen nach Schlüsseln müssen wiederholt Hashes berechnet und die gefundenen Elemente geprüft werden.

Beispiel: Doppel-Hashing

- ▶ Weiche bei Kollisionen auf eine zweite Hashfunktion aus.
- ▶ Multipliziere die Hash-Werte mit der Anzahl der Versuche.

Beispiel: Kuckucks-Hashing

- ▶ Verwende zwei Hashtabellen mit zwei Hashfunktionen.
- ▶ Verdränge bei Kollisionen ggf. das vorgefundene Element
- ▶ Füge verdrängte Elemente in die andere Hashtabelle ein.

Hashfunktionen

Zusammenfassung

- ▶ Speichere Elemente in einer Hashmap, bei der Positionen berechnet werden.
- ▶ Verwende Hashfunktion mit möglichst wenigen Kollisionen.
- ▶ Verwende viel Speicher, um Kollisionen zu vermeiden.
- ▶ Bei Kollisionen verwende finde eine neue Position oder speichere Elemente in Listen.

Hashfunktionen

Zusammenfassung

- ▶ Speichere Elemente in einer Hashmap, bei der Positionen berechnet werden.
- ▶ Verwende Hashfunktion mit möglichst wenigen Kollisionen.
- ▶ Verwende viel Speicher, um Kollisionen zu vermeiden.
- ▶ Bei Kollisionen verwende finde eine neue Position oder speichere Elemente in Listen.

Eigenschaften von Hashmaps

- ▶ **Effizienz**: Hashfunktion berechnet schnell Positionen.
- ▶ Average Case: $O(1)$ für Suchen und Einfügen.
- ▶ Worst Case: $O(n)$ für Suchen und Einfügen.
 - ▶ geschlossenes Hashing: Ggf. viele Berechnungen notwendig.
 - ▶ offenes Hashing: Ggf. lange Listen an wenigen Positionen.