Wiederholung

22. Juli 2016

Hashing

Hashing

Beispiel Hashmap

Hashing

Beispiel Hashmap

- Wir wollen eine große Menge (Universum) auf eine kleinere abbilden

Hashing

Beispiel Hashmap

- Wir wollen eine große Menge (Universum) auf eine kleinere abbilden
- Z.b. Telefonbuch: (Name, Telefonnummer)-Paare

Hashing

Beispiel Hashmap

- Wir wollen eine große Menge (Universum) auf eine kleinere abbilden
- Z.b. Telefonbuch: (Name, Telefonnummer)-Paare
- 100 Einträge in einem Array speichern

Hashing

Beispiel Hashmap

- Wir wollen eine große Menge (Universum) auf eine kleinere abbilden
- Z.b. Telefonbuch: (Name, Telefonnummer)-Paare
- 100 Einträge in einem Array speichern
- Universum sind alle Strings (selbst bei max length < 30 gigantisch viele)

Hashing

Beispiel Hashmap

- Wir wollen eine große Menge (Universum) auf eine kleinere abbilden
- Z.b. Telefonbuch: (Name, Telefonnummer)-Paare
- 100 Einträge in einem Array speichern
- Universum sind alle Strings (selbst bei max length < 30 gigantisch viele)
- Wollen Array der Länge 100 benutzen

Hashing

Beispiel Hashmap

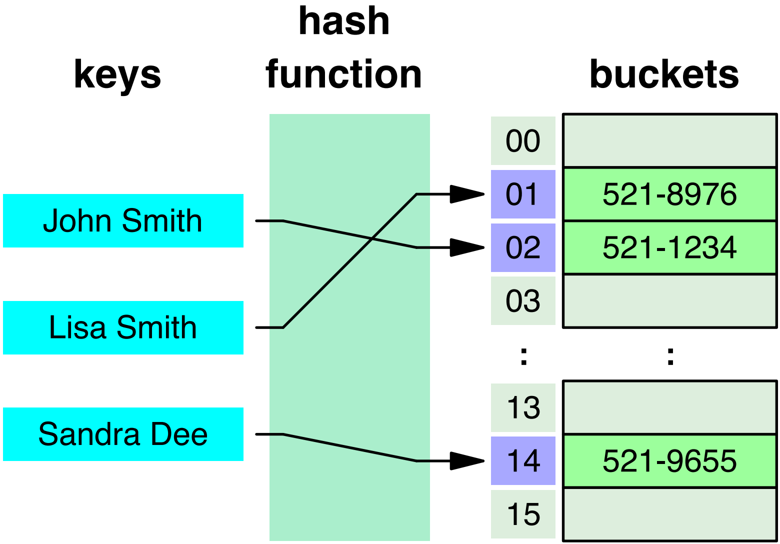
- Wir wollen eine große Menge (Universum) auf eine kleinere abbilden
- Z.b. Telefonbuch: (Name, Telefonnummer)-Paare
- 100 Einträge in einem Array speichern
- Universum sind alle Strings (selbst bei max length < 30 gigantisch viele)
- Wollen Array der Länge 100 benutzen
- Wie mappen wir den Namen zum Array Index?

Hashing

Beispiel Hashmap

- Wir wollen eine große Menge (Universum) auf eine kleinere abbilden
- Z.b. Telefonbuch: (Name, Telefonnummer)-Paare
- 100 Einträge in einem Array speichern
- Universum sind alle Strings (selbst bei max length < 30 gigantisch viele)
- Wollen Array der Länge 100 benutzen
- Wie mappen wir den Namen zum Array Index?
  $\rightarrow$ Hashfunktion

Hashing



hash
function
keys
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
:
13
14
15
521-8976
521-1234
:
521-9655

Hashing

- Hashfunktion $h : U \to \{1, \ldots, m\}$

Hashing

- Hashfunktion $h : U \to \{1, \ldots, m\}$
- $h(x)$ ist dann der Index im Array von x

Hashing

- Hashfunktion $h : U \to \{1, \ldots, m\}$
- $h(x)$ ist dann der Index im Array von x
- Angenommen wir wollen (x, y) im Array a speichern. x ist der Schlüssel (Name) und y der Wert (Telefonummer)

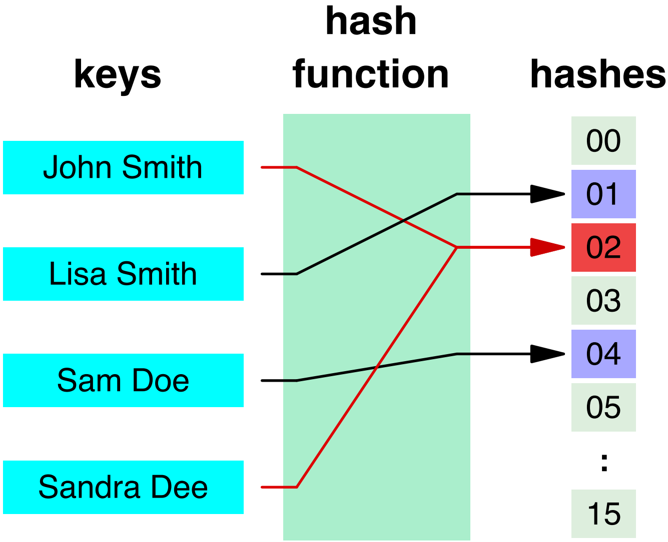
Hashing

- Hashfunktion $h : U \to \{1, \ldots, m\}$
- $h(x)$ ist dann der Index im Array von x
- Angenommen wir wollen (x, y) im Array a speichern. x ist der Schlüssel (Name) und y der Wert (Telefonummer)
- Setze $a[h(x)] = y$

Hashing

- Hashfunktion $h : U \to \{1, \dots, m\}$
- $h(x)$ ist dann der Index im Array von x
- Angenommen wir wollen (x, y) im Array a speichern. x ist der Schlüssel (Name) und y der Wert (Telefonummer)
- Setze $a[h(x)] = y$
- Nummer von x auslesen: $a[h(x)]$ zurückgeben

Hashing


hash
function
keys
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
15

Hashing: Kollisionen

Eine einfache Möglichkeit:

Hashing: Kollisionen

Eine einfache Möglichkeit:

- Die Hashtabelle hat m Positionen

Hashing: Kollisionen

Eine einfache Möglichkeit:

- Die Hashtabelle hat m Positionen
- Jede Position besteht aus einer verketteten Liste

Hashing: Kollisionen

Eine einfache Möglichkeit:

- Die Hashtabelle hat m Positionen
- Jede Position besteht aus einer verketteten Liste
- Die Liste auf Position k enthält alle x mit $h(x) = k$

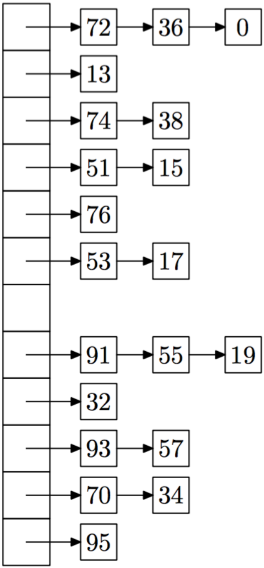
Hashing: Kollisionen

Eine einfache Möglichkeit:

- Die Hashtabelle hat m Positionen
- Jede Position besteht aus einer verketteten Liste
- Die Liste auf Position k enthält alle x mit $h(x) = k$

Vorteile: Einfügen, Suchen und Löschen sehr einfach.
Nachteile: Speicherverbrauch (Overhead durch Listen)

Hashing



Hashfunktionen

Was sind Hashfunktionen?

Hashfunktionen

Was sind Hashfunktionen?
Im Prinzip alle Funktionen der Form $h : U \to \{1, \ldots, m\}$

Hashfunktionen

Was sind Hashfunktionen?
Im Prinzip alle Funktionen der Form $h : U \to \{1, \ldots, m\}$
Am liebsten wollen wir injektives h, d.h. $h(x) \neq h(y)$ für alle $x \neq y \in S$ wobei $S \subset U$ die Schlüssel sind die wir speichern wollen. Dann keine Kollisionen.

Hashfunktionen

Was sind Hashfunktionen?
Im Prinzip alle Funktionen der Form $h : U \to \{1, \ldots, m\}$
Am liebsten wollen wir injektives h, d.h. $h(x) \neq h(y)$ für alle $x \neq y \in S$ wobei $S \subset U$ die Schlüssel sind die wir speichern wollen. Dann keine Kollisionen.
Darauf können wir aber in der Praxis nicht hoffen, da es zu stark von S abhängt

Universelles Hashing

Definition

Es sei H eine nicht-leere Menge von Funktionen $U \to \{1, \ldots, m\}$. Wir sagen, dass H eine *universelle Familie von Hashfunktionen* ist, wenn für jedes $x, y \in U$, $x \neq y$ folgendes gilt:

$$\frac{|h \in H \mid h(x) = h(y)|}{|H|} \leq \frac{1}{m}$$

Universelles Hashing

Sei $U = \{0, \dots, p-1\}$, wobei p eine Primzahl ist. Es sei $h_{a,b}(x) = ((ax + b) \mod p) \mod m$. Wir definieren

$$H = \{h_{a,b} \mid 1 \leq a < p, 0 \leq b < p\}$$

Theorem

H ist eine universelle Familie von Hashfunktionen.

Hashing

Hashen von komplexeren Typen als Zahlen, wie z.B. Strings oder Java-Objekten ist nicht trivial!

Hashing

Hashen von komplexeren Typen als Zahlen, wie z.B. Strings oder Java-Objekten ist nicht trivial!
In der Vorlesung nicht im Detail behandelt, sollte man aber im Hinterkopf haben.

Hashing

Hashen von komplexeren Typen als Zahlen, wie z.B. Strings oder Java-Objekten ist nicht trivial!
In der Vorlesung nicht im Detail behandelt, sollte man aber im Hinterkopf haben.
Viel benutzte Hashfunktionen: MD5, SHA1, SHA2

Weitere Anwendungen

- Vermeiden Passwörter zu speichern in Datenbanken, z.B. auf Webseiten

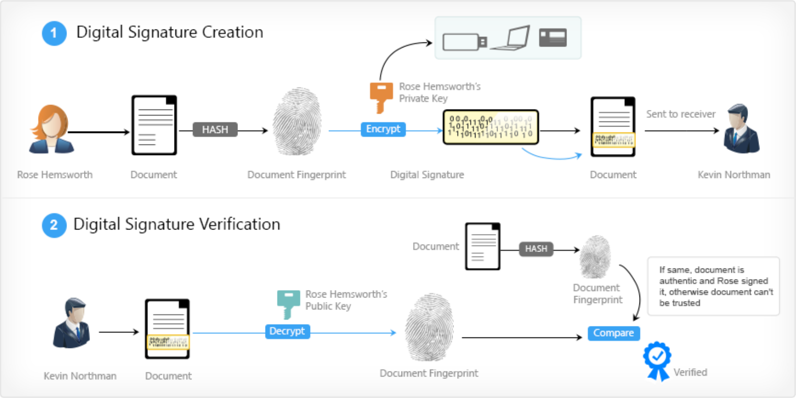
Weitere Anwendungen

- Vermeiden Passwörter zu speichern in Datenbanken, z.B. auf Webseiten
- Schneller filter für gleiche Elemente (hashing kann schneller sein als kompletter vergleich)

Weitere Anwendungen

- Vermeiden Passwörter zu speichern in Datenbanken, z.B. auf Webseiten
- Schneller filter für gleiche Elemente (hashing kann schneller sein als kompletter vergleich)
- Digitale Signaturen

Digitale Signatur



Quelle: signinghub.com

Union-Find

Union-Find

Wir wollen eine Menge von disjunkten Mengen speichern, so dass

Union-Find

Wir wollen eine Menge von disjunkten Mengen speichern, so dass

- wir schnell überprüfen können ob zwei Elemente zur selben Menge gehören und

Union-Find

Wir wollen eine Menge von disjunkten Mengen speichern, so dass

- wir schnell überprüfen können ob zwei Elemente zur selben Menge gehören und
- wir schnell zwei Mengen vereinigen können.

Union-Find

Wir wollen eine Menge von disjunkten Mengen speichern, so dass

- wir schnell überprüfen können ob zwei Elemente zur selben Menge gehören und
- wir schnell zwei Mengen vereinigen können.

Lösung: Union-find Datenstruktur

Union-Find: naiv

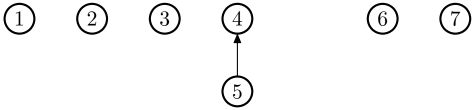
union(a, b): Hänge *find*(a) bei *find*(b) ein

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$...

Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

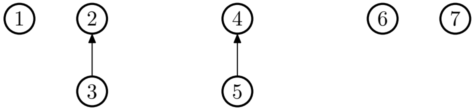
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)\ldots$



Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$...



Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$...

Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$...

Union-Find: union by rank

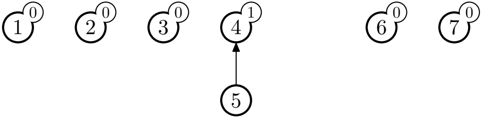
union(a, b): Verwende die ranghöhere Wurzel

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$

Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

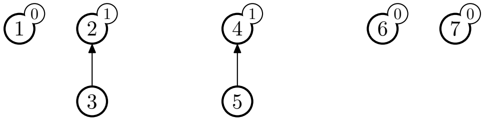
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

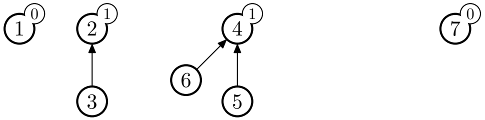
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

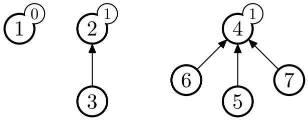
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

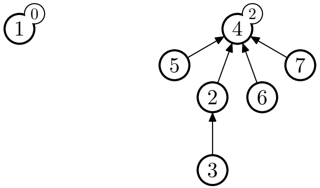
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

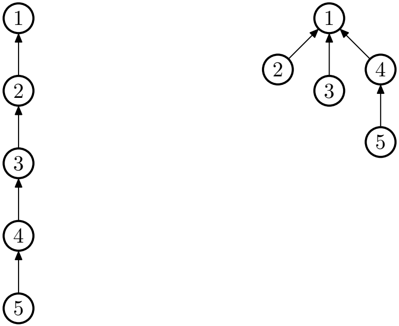
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: Pfadkompression

find(a): Komprimiere durchlaufene Pfade

Beispiel: *find*(4)



Union-Find

Anwendungsgebiete

Union-Find

Anwendungsgebiete

- Buchführen über Graphkomponenten (z.B. Spannbaum mit Kruskal)

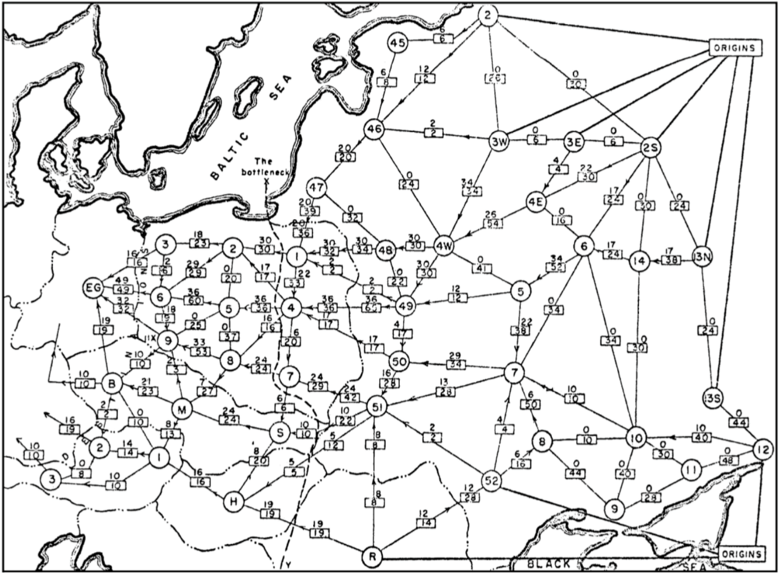
Union-Find

Anwendungsgebiete

- Buchführen über Graphkomponenten (z.B. Spannbaum mit Kruskal)
- Unifikation von logischen Formeln (z.B. in Prolog)

Anwendung von Min-Cut

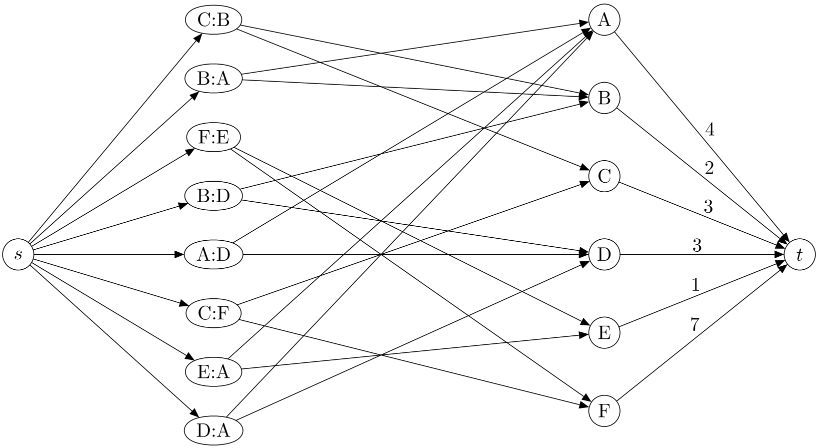
Anwendung Min-Cut



Anwendung Min-Cut

Eine Anwendung von Min-Cut ist auch immer eine Anwendung von Max-Flow.

Anwendung Max-Flow



Weitere Anwendung Max-Flow

Weitere Anwendung Max-Flow

- Angebot und Nachfrage in Transportnetzwerk

Weitere Anwendung Max-Flow

- Angebot und Nachfrage in Transportnetzwerk
- Bipartitetes Matching

Weitere Anwendung Max-Flow

- Angebot und Nachfrage in Transportnetzwerk
- Bipartitetes Matching
- Image Segmentation

Weitere Anwendung Max-Flow

- Angebot und Nachfrage in Transportnetzwerk
- Bipartitetes Matching
- Image Segmentation
- Dichtester Untergraph

Finden eines Min-Cut

Min-Cut Max-Flow

Theorem

Sei f ein Fluß im s-t-Netzwerk $G = (V, E)$. Dann sind äquivalent:

Min-Cut Max-Flow

Theorem

Sei f ein Fluß im s-t-Netzwerk $G = (V, E)$. Dann sind äquivalent:

1. *f ist ein maximaler Fluss*

Min-Cut Max-Flow

Theorem

Sei f ein Fluß im s-t-Netzwerk $G = (V, E)$. Dann sind äquivalent:

1. *f ist ein maximaler Fluss*
2. *In G_f gibt es keinen augmentierenden Pfad*

Min-Cut Max-Flow

Theorem

Sei f ein Fluß im s-t-Netzwerk $G = (V, E)$. Dann sind äquivalent:

1. *f ist ein maximaler Fluss*
2. *In G_f gibt es keinen augmentierenden Pfad*
3. *$|f| = c$ wobei c die Kapazität des minimalen Schnittes in G ist*

Minimalen schnitt bestimmen

Angenommen wir haben den maximalen Fluss f und Residualnetz G_f.

Minimalen schnitt bestimmen

Angenommen wir haben den maximalen Fluss f und Residualnetz G_f.

- Sei S die Menge aller von s erreichbaren Knoten in G_f

Minimalen schnitt bestimmen

Angenommen wir haben den maximalen Fluss f und Residualnetz G_f.

- Sei S die Menge aller von s erreichbaren Knoten in G_f
- Sei $T = V - S$ die restlichen Knoten

Minimalen schnitt bestimmen

Angenommen wir haben den maximalen Fluss f und Residualnetz G_f.

- Sei S die Menge aller von s erreichbaren Knoten in G_f
- Sei $T = V - S$ die restlichen Knoten
- (S, T) ist ein minimaler Schnitt

Minimalen schnitt bestimmen

Angenommen wir haben den maximalen Fluss f und Residualnetz G_f.

- Sei S die Menge aller von s erreichbaren Knoten in G_f
- Sei $T = V - S$ die restlichen Knoten
- (S, T) ist ein minimaler Schnitt

Achtung: Nur die Kanten von S nach T werden zum Schnitt gezählt, nicht die von T nach S!