Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Algorithmen und Datenstrukturen

Vorlesung 12:
Hashing

Alexander Wolff

Wintersemester 2024

Hashing

Wörterbuch.

- Spezialfall einer dynamischen Menge
- Anwendung: im Compiler Symboltabelle für Schlüsselwörter

Abstrakter Datentyp.

stellt folgende Operationen bereit: INSERT, DELETE, SEARCH

Implementierung.

Hashing

Wörterbuch.

- Spezialfall einer dynamischen Menge
- Anwendung: im Compiler Symboltabelle für Schlüsselwörter

Abstrakter Datentyp.

stellt folgende Operationen bereit: INSERT, DELETE, SEARCH

Implementierung.

heute: Hashing

Hashing

Wörterbuch.

- Spezialfall einer dynamischen Menge
- Anwendung: im Compiler Symboltabelle für Schlüsselwörter

Abstrakter Datentyp.

stellt folgende Operationen bereit: INSERT, DELETE, SEARCH

Implementierung.

heute: Hashing

engl. to hash = zerhacken, kleinschneiden

Hashing

Wörterbuch.

- Spezialfall einer dynamischen Menge
- Anwendung: im Compiler Symboltabelle für Schlüsselwörter

Abstrakter Datentyp.

stellt folgende Operationen bereit: INSERT, DELETE, SEARCH

Implementierung.

heute: Hashing

engl. to hash = zerhacken, kleinschneiden

Suchzeit:

Hashing

Wörterbuch.

- Spezialfall einer dynamischen Menge
- Anwendung: im Compiler Symboltabelle für Schlüsselwörter

Abstrakter Datentyp.

stellt folgende Operationen bereit: INSERT, DELETE, SEARCH

Implementierung.

heute: Hashing

engl. to hash = zerhacken, kleinschneiden

Suchzeit:

- im schlechtesten Fall $\Theta(n)$,

Hashing

Wörterbuch.

- Spezialfall einer dynamischen Menge
- Anwendung: im Compiler Symboltabelle für Schlüsselwörter

Abstrakter Datentyp.

stellt folgende Operationen bereit: INSERT, DELETE, SEARCH

Implementierung.

heute: Hashing

engl. to hash = zerhacken, kleinschneiden

Suchzeit:

- im schlechtesten Fall $\Theta(n)$,
- erwartet $\mathcal{O}(1)$ unter akzeptablen Annahmen

Direkte Adressierung

Direkte Adressierung

Annahmen.

Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$

Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

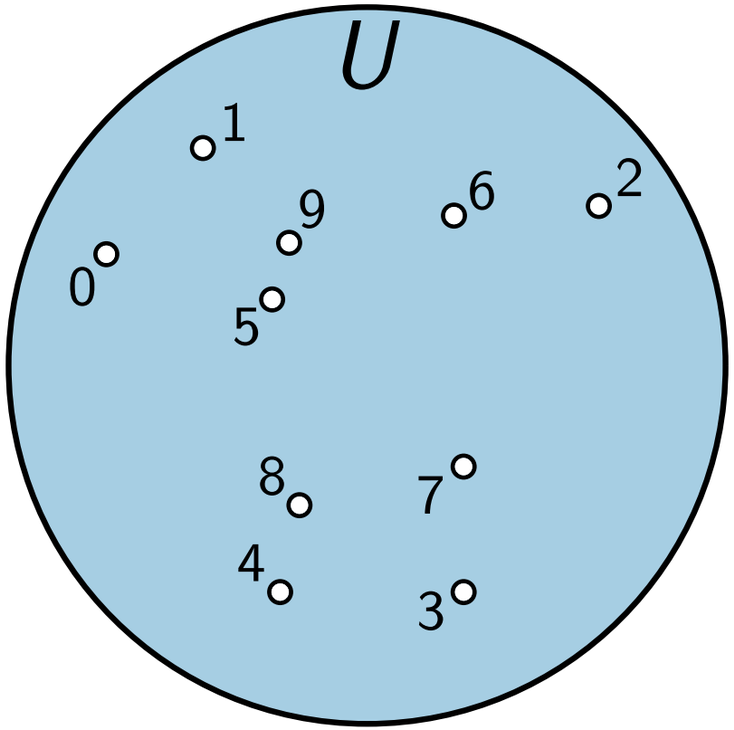
Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Universum



Direkte Adressierung

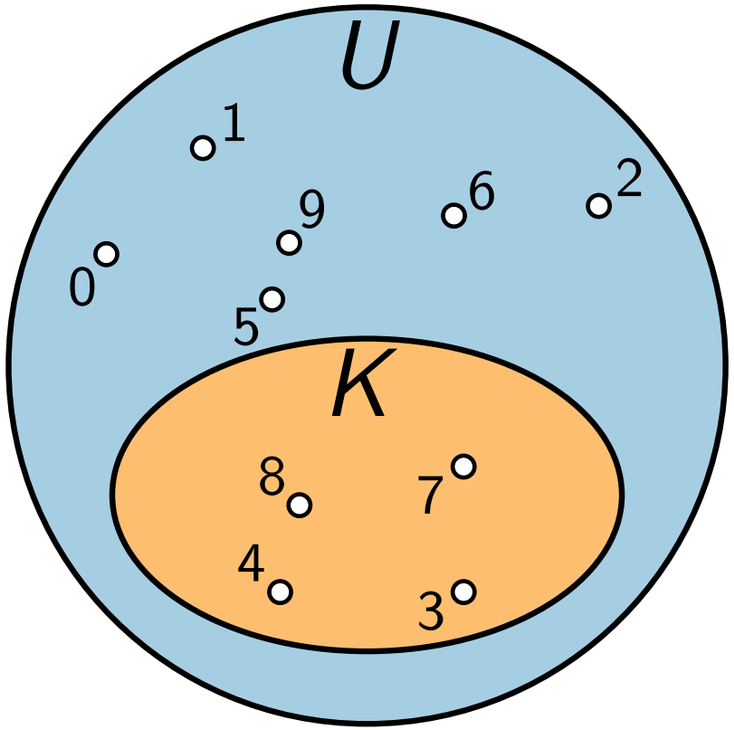
Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Universum

Menge der momentan benutzten Schlüssel



Direkte Adressierung

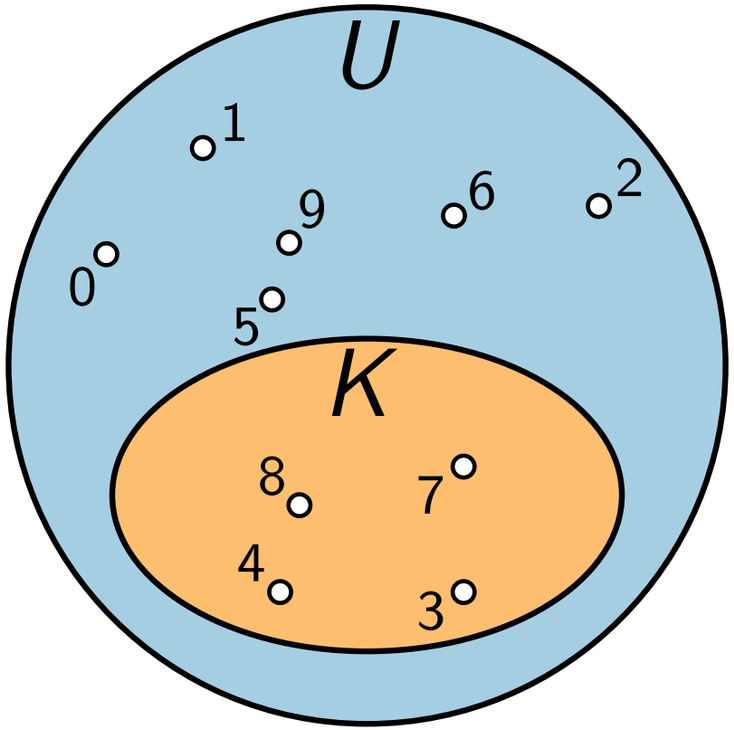
Annahmen.

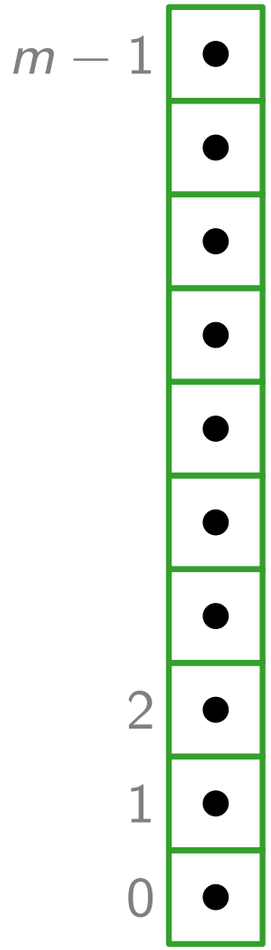
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Universum

Menge der momentan benutzten Schlüssel





Direkte Adressierung

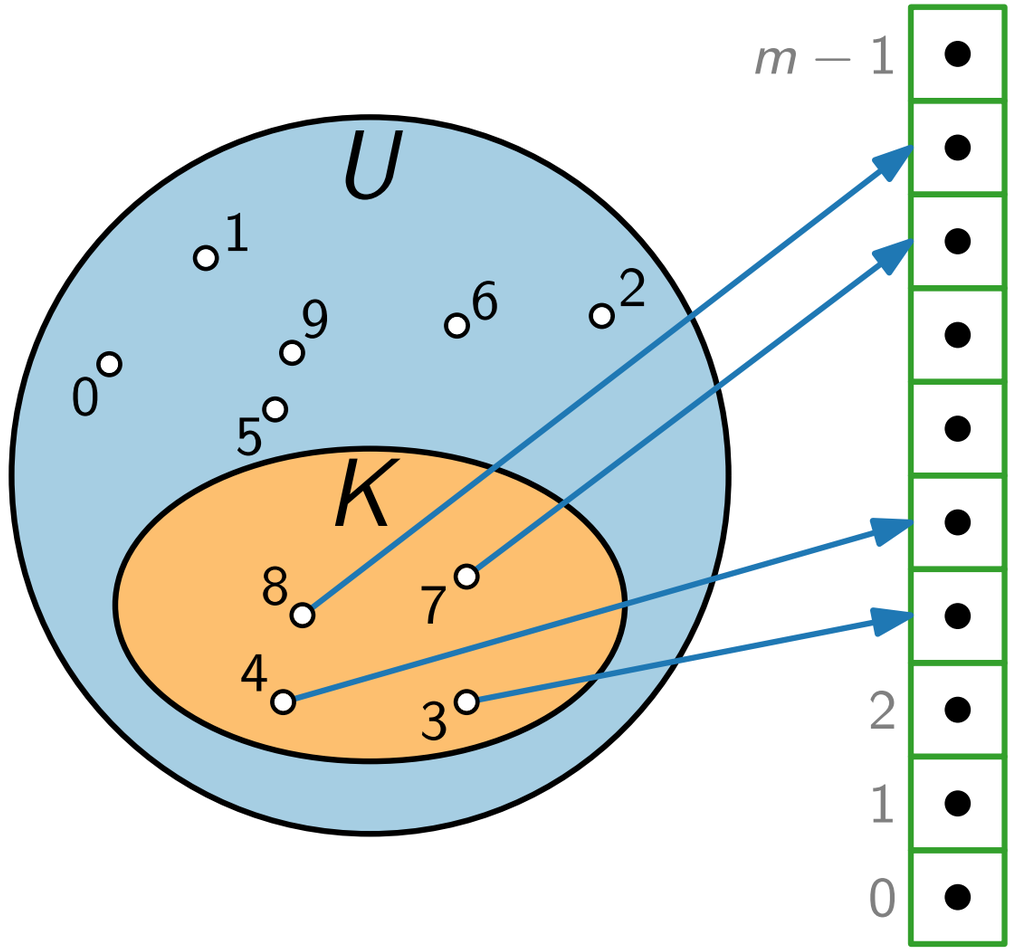
Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Universum

Menge der momentan benutzten Schlüssel

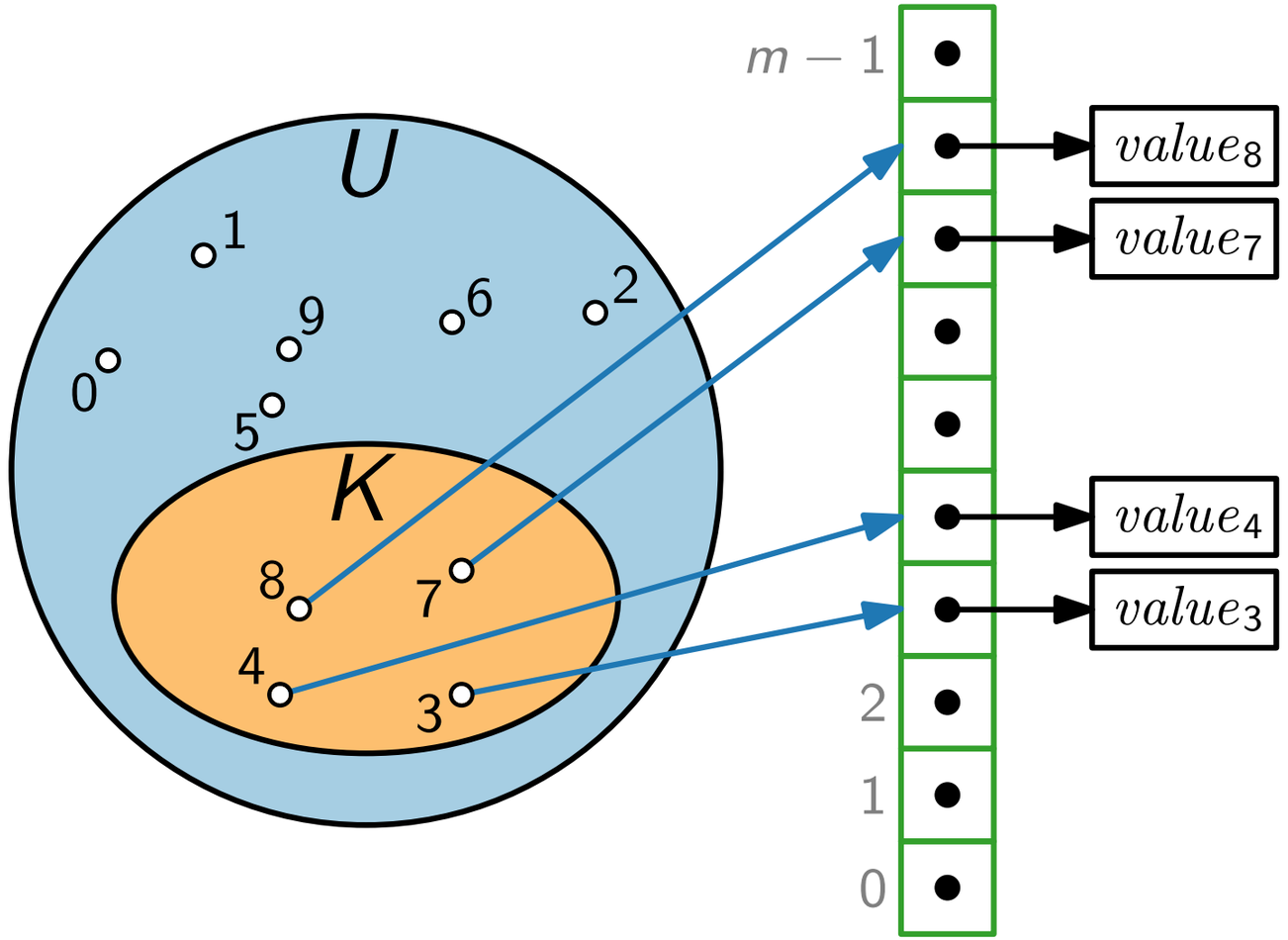


Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!



Direkte Adressierung

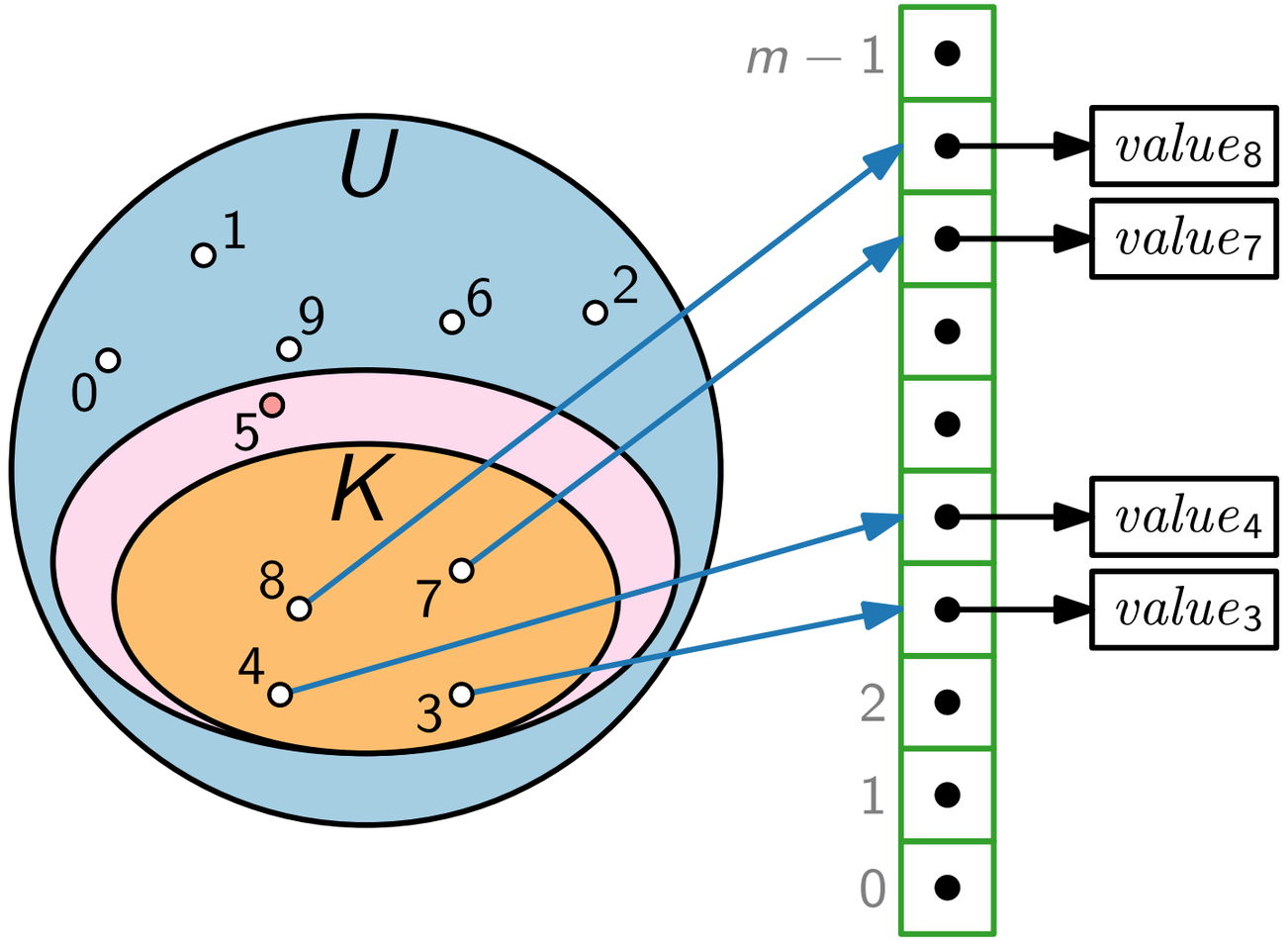
Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Universum

Menge der momentan benutzten Schlüssel



Direkte Adressierung

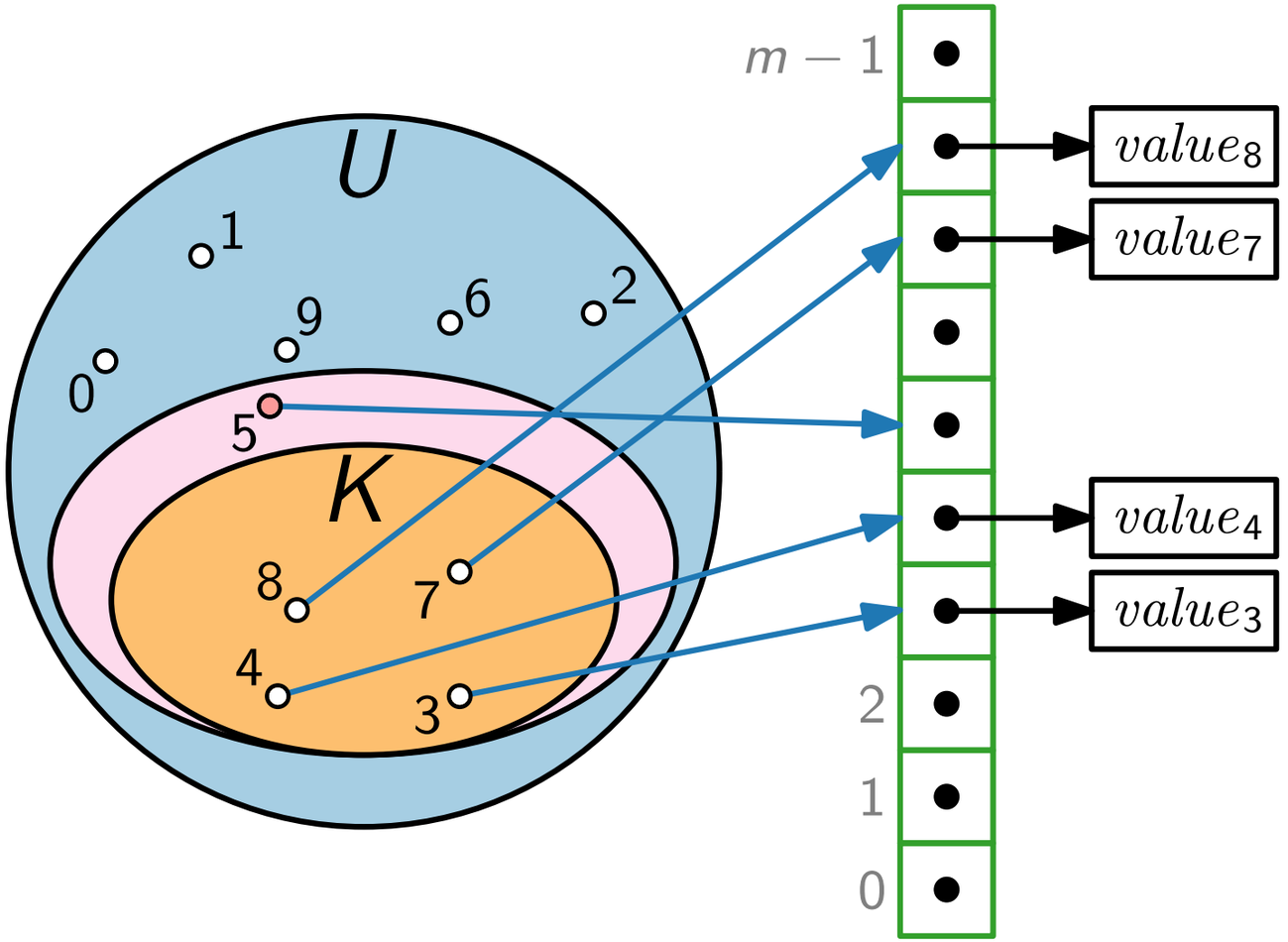
Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Universum

Menge der momentan benutzten Schlüssel



Direkte Adressierung

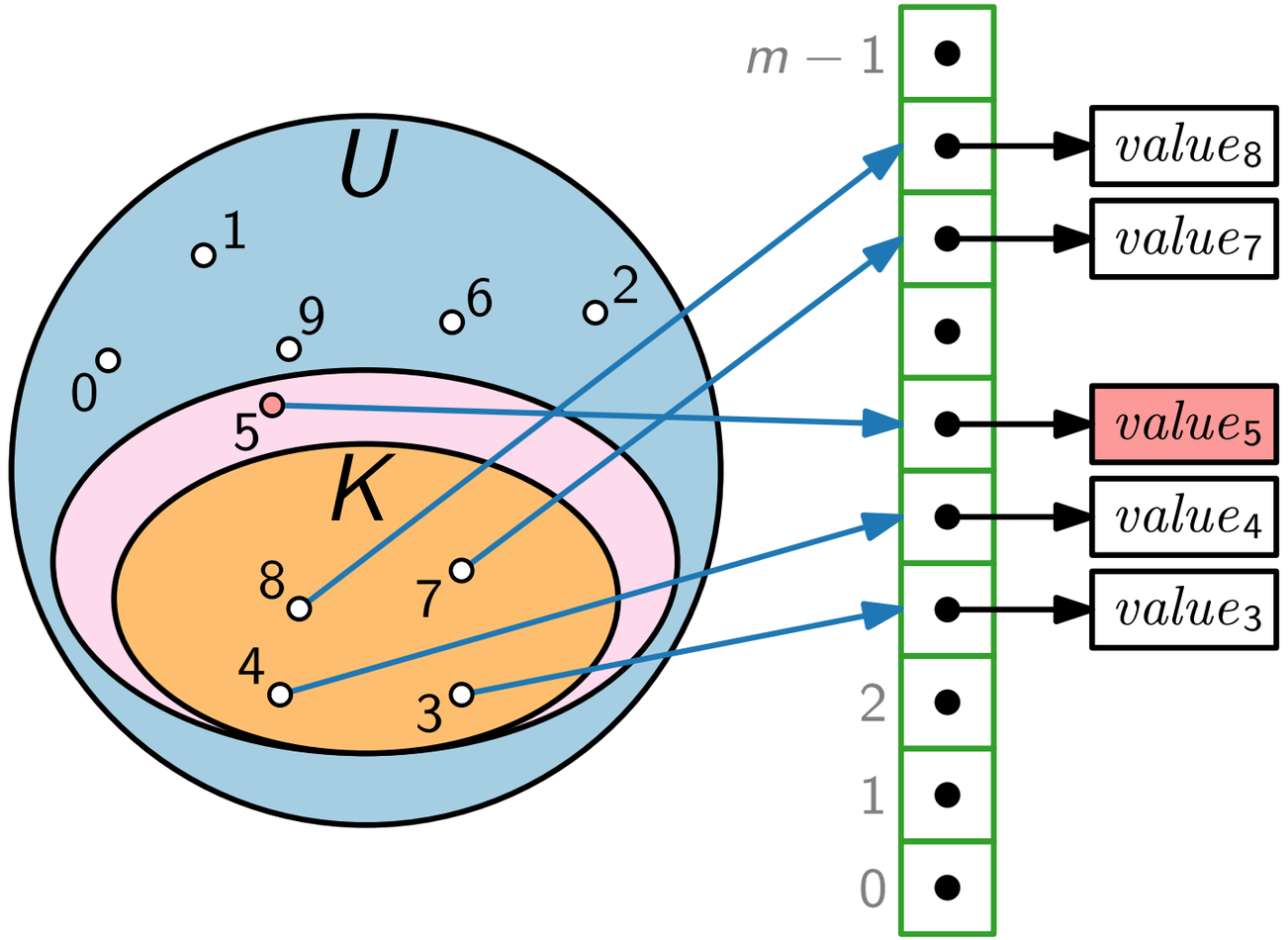
Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Universum

Menge der momentan benutzten Schlüssel



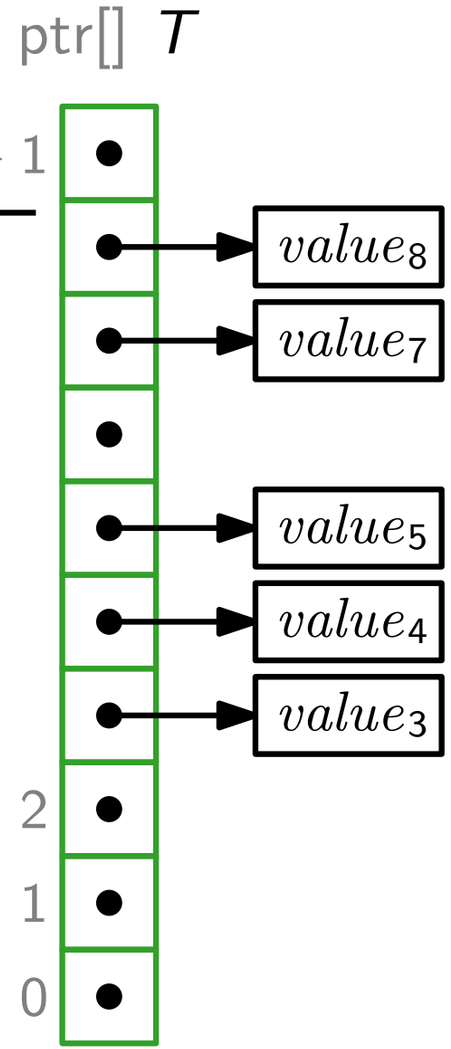
Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**



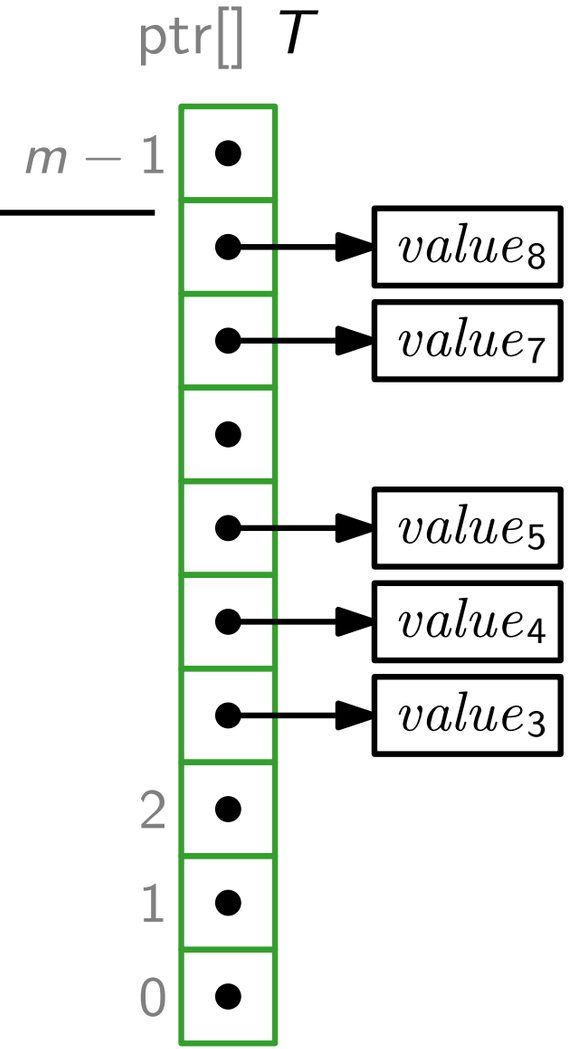
Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	
ptr Insert(key k, value v)	
Delete(key k)	
ptr Search(key k)	



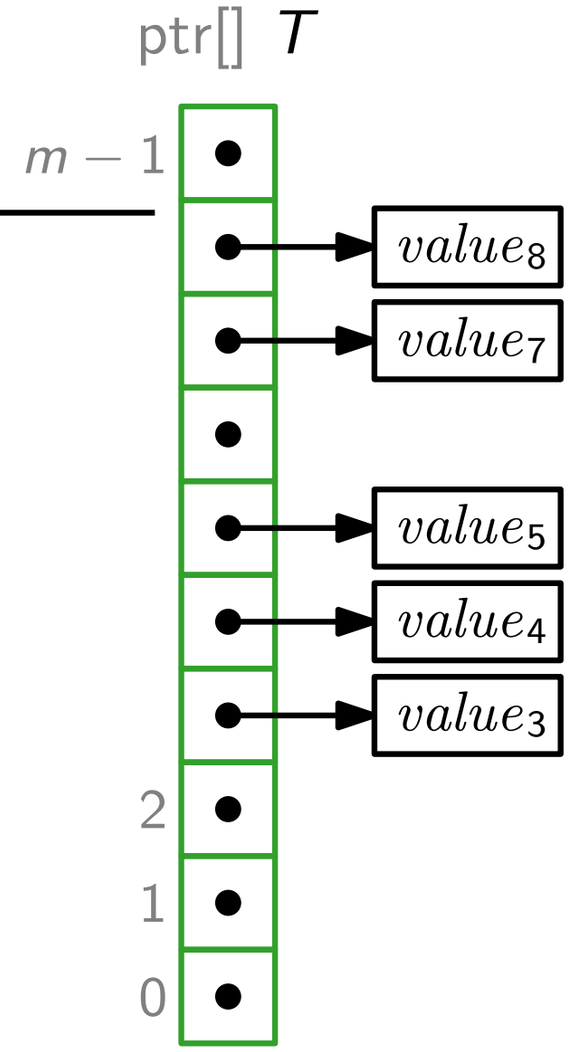
Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	
Konstruktor	
ptr Insert(key k, value v)	
Delete(key k)	
ptr Search(key k)	



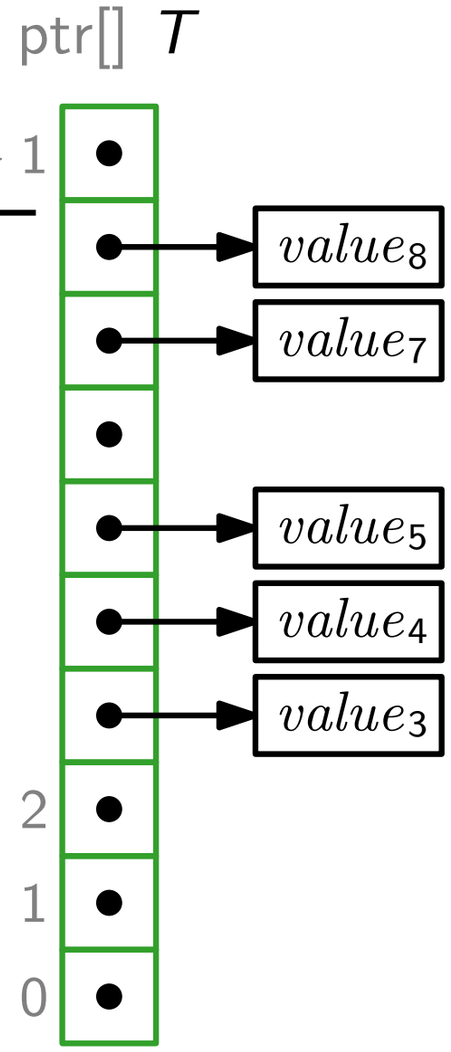
Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	$T =$ **new** value$[0 \ldots m-1]$
ptr Insert(key k, value v)	
Delete(key k)	
ptr Search(key k)	



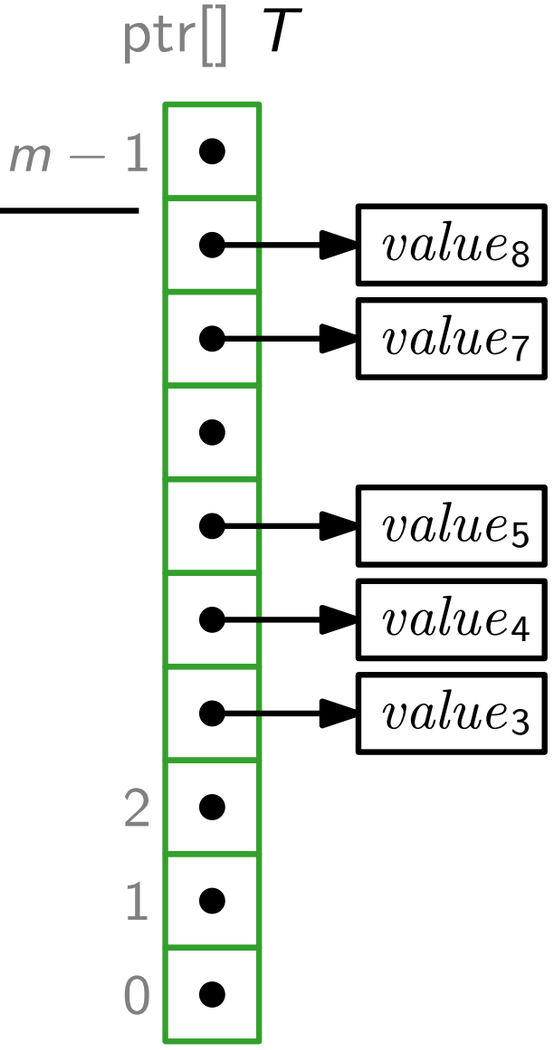
Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$
ptr Insert(key k, value v)	
Delete(key k)	
ptr Search(key k)	



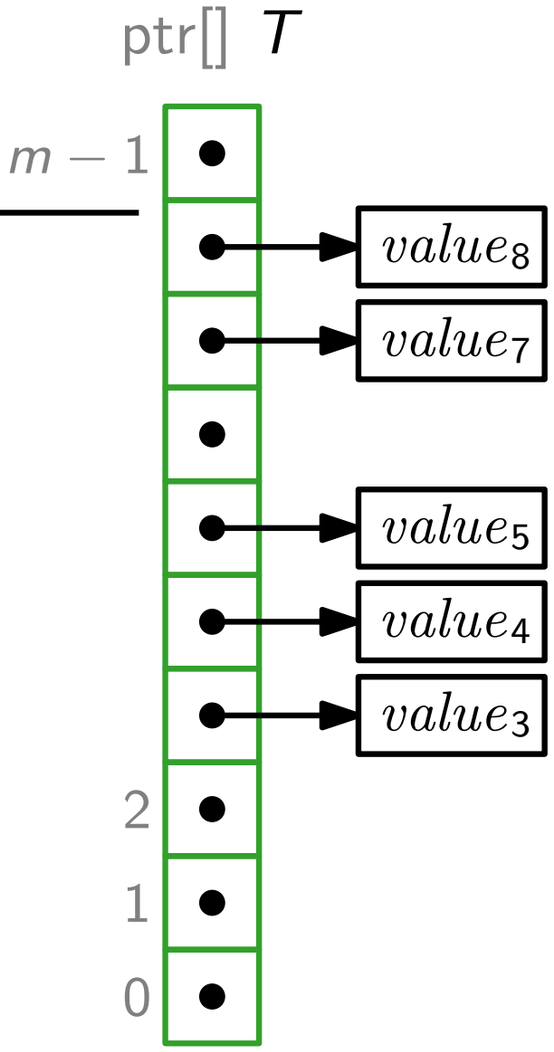
Direkte Adressierung

Annahmen.

- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz
ptr Insert(key k, value v)	
Delete(key k)	
ptr Search(key k)	

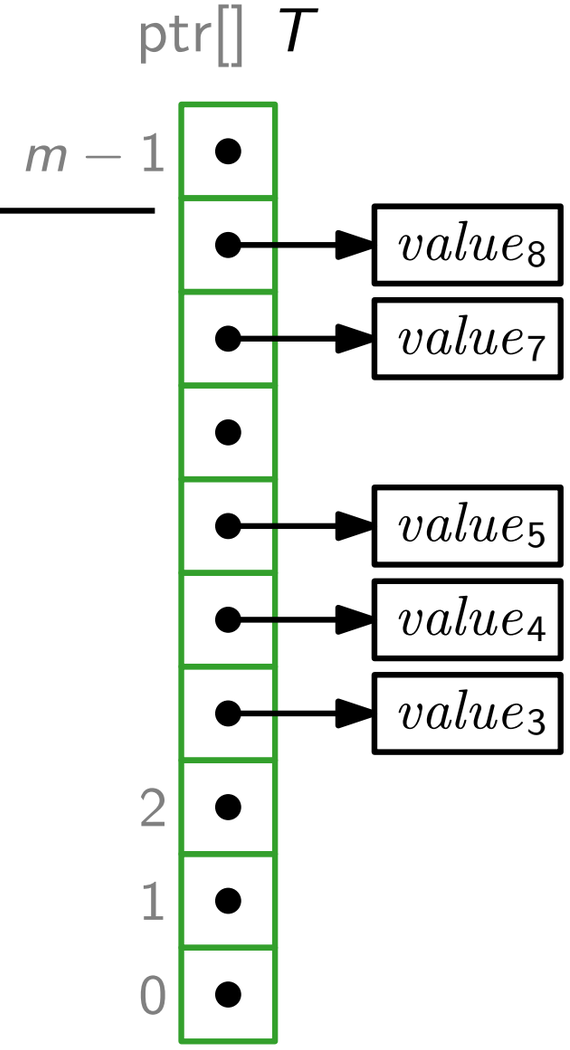


Direkte Adressierung

Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz
ptr Insert(key k, value v)	// lege neuen Datensatz an // und initialisiere ihn mit v
Delete(key k)	
ptr Search(key k)	



Direkte Adressierung

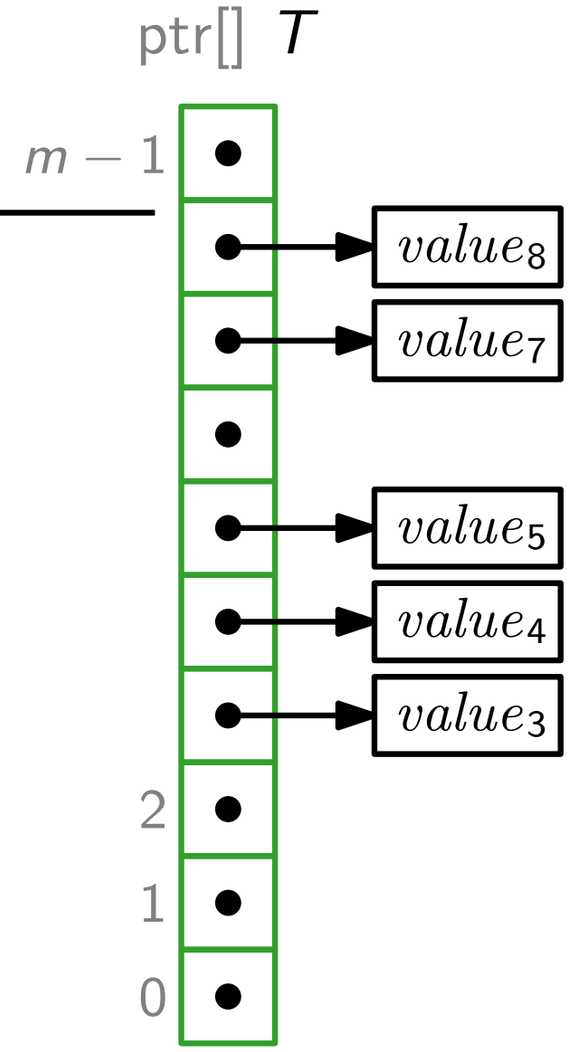
Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz

ptr Insert(key k, value v)	// lege neuen Datensatz an
	// und initialisiere ihn mit v
	$T[k]$ = **new** value(v)

Delete(key k)	
ptr Search(key k)	



Direkte Adressierung

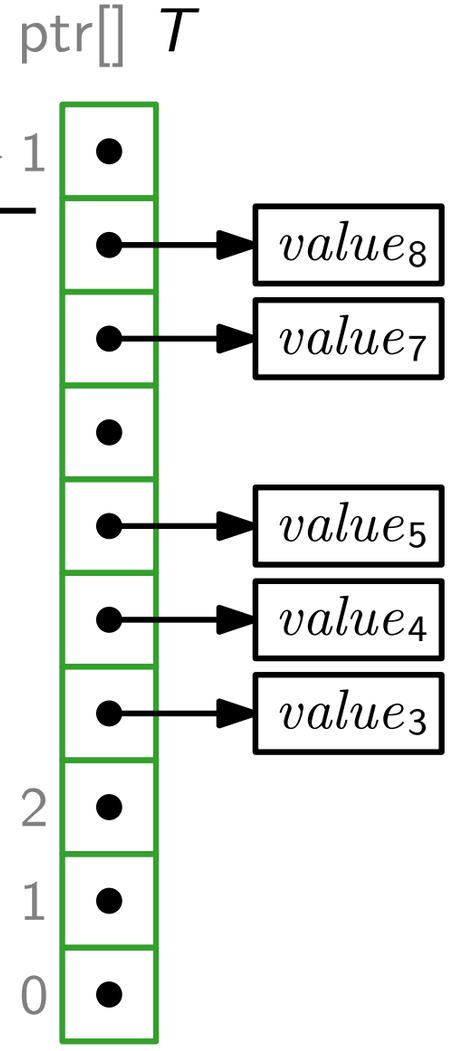
Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz

ptr Insert(key k, value v)	// lege neuen Datensatz an
	// und initialisiere ihn mit v
	$T[k]$ = **new** value(v)

Delete(key k)	// Speicher freigeben, auf den $T[k]$ zeigt
ptr Search(key k)	



Direkte Adressierung

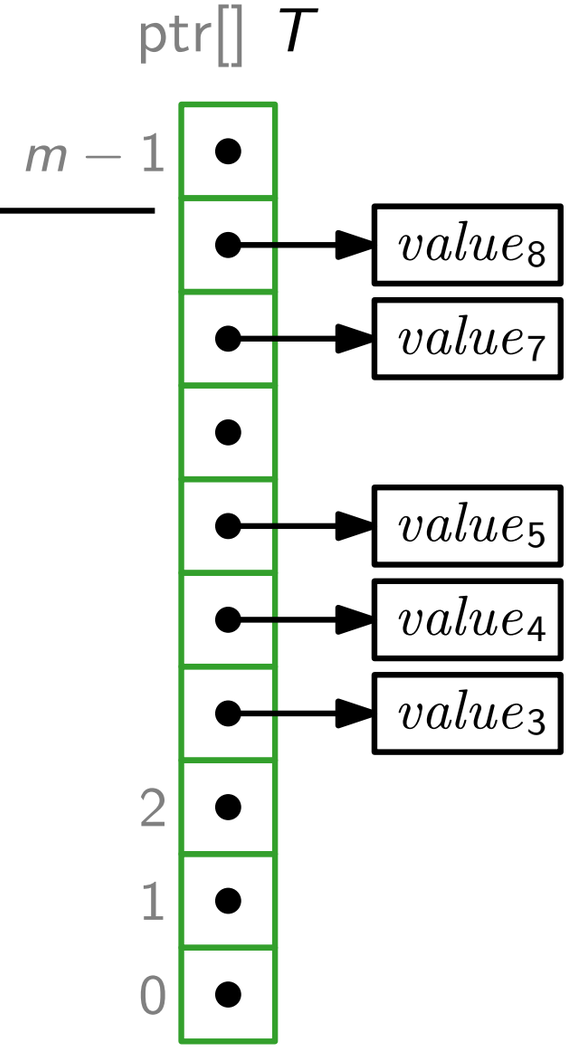
Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz

ptr Insert(key k, value v)	// lege neuen Datensatz an
	// und initialisiere ihn mit v
	$T[k]$ = **new** value(v)

Delete(key k)	// Speicher freigeben, auf den $T[k]$ zeigt $T[k] = nil$
ptr Search(key k)	



Direkte Adressierung

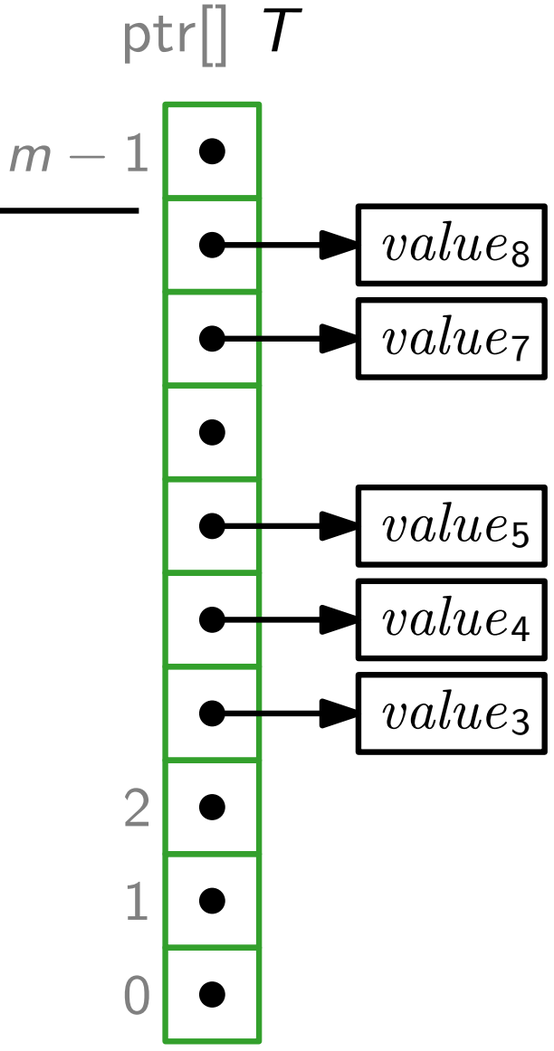
Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz

ptr Insert(key k, value v)	// lege neuen Datensatz an
	// und initialisiere ihn mit v
	$T[k]$ = **new** value(v)

Delete(key k)	// Speicher freigeben, auf den $T[k]$ zeigt $T[k] = nil$
ptr Search(key k)	**return** $T[k]$



Direkte Adressierung

Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

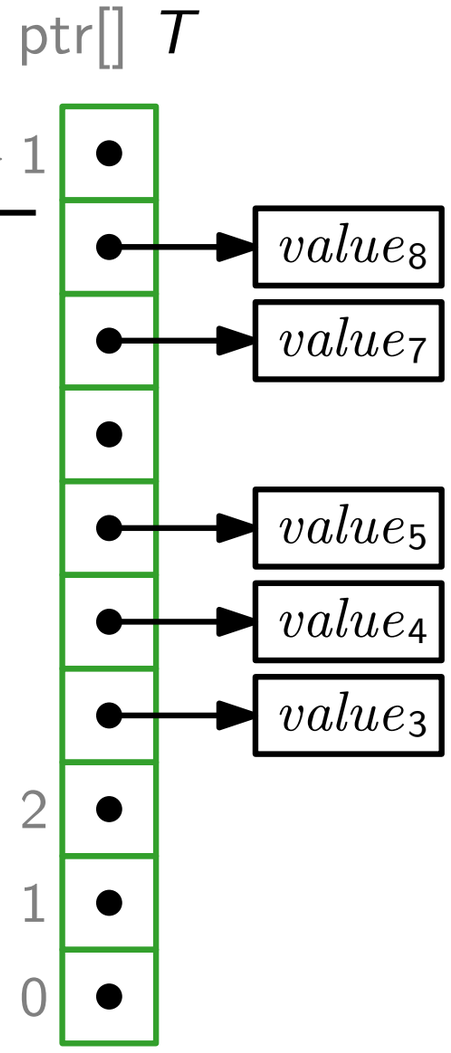
dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz

ptr Insert(key k, value v)	// lege neuen Datensatz an
	// und initialisiere ihn mit v
	$T[k]$ = **new** value(v)

Delete(key k)	// Speicher freigeben, auf $T[k] = nil$
ptr Search(key k)	**return** $T[k]$

Laufzeiten?



Direkte Adressierung

Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

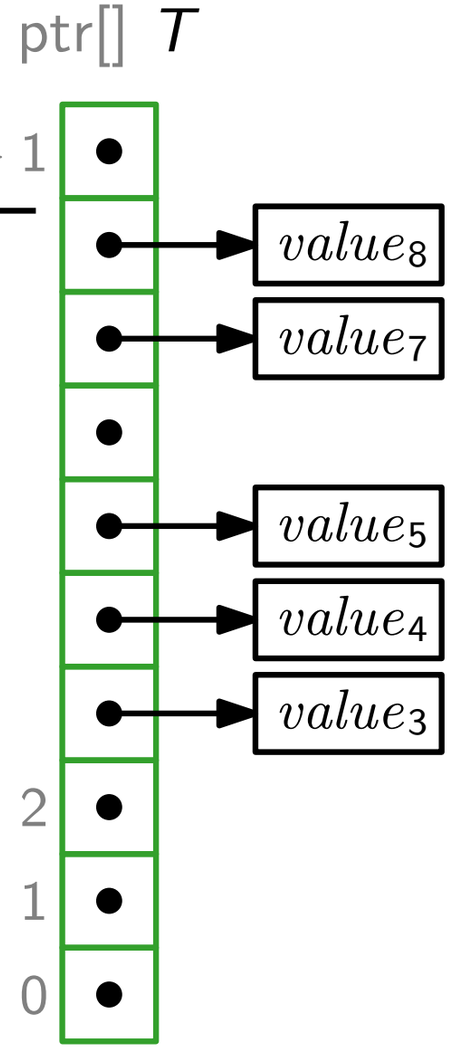
dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz

ptr Insert(key k, value v)	// lege neuen Datensatz an
	// und initialisiere ihn mit v
	$T[k]$ = **new** value(v)

Delete(key k)	// Speicher freigeben, auf $T[k] = nil$
ptr Search(key k)	**return** $T[k]$

Laufzeiten?
Insert, Delete, Search $\mathcal{O}(1)$ im *schlechtesten Fall*



Direkte Adressierung

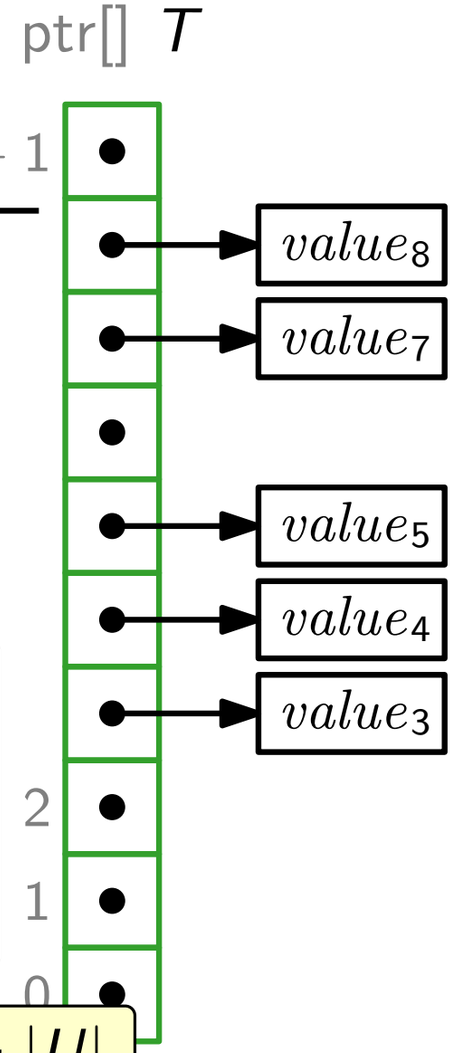
Annahmen.
- Schlüssel aus kleinem **Universum** $U = \{0, \ldots, m-1\}$
- Schlüssel paarweise verschieden

dyn. Menge!

Operation	**Implementierung**
HashDA(int m)	T = **new** value$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = nil$ // $T[j]$ = Zeiger auf j. Datensatz

ptr Insert(key k, value v)	// lege neuen Datensatz an
	// und initialisiere ihn mit v
	$T[k]$ = **new** value(v)

Delete(key k)	// Speicher freigeben, auf $T[k] = nil$
ptr Search(key k)	**return** $T[k]$

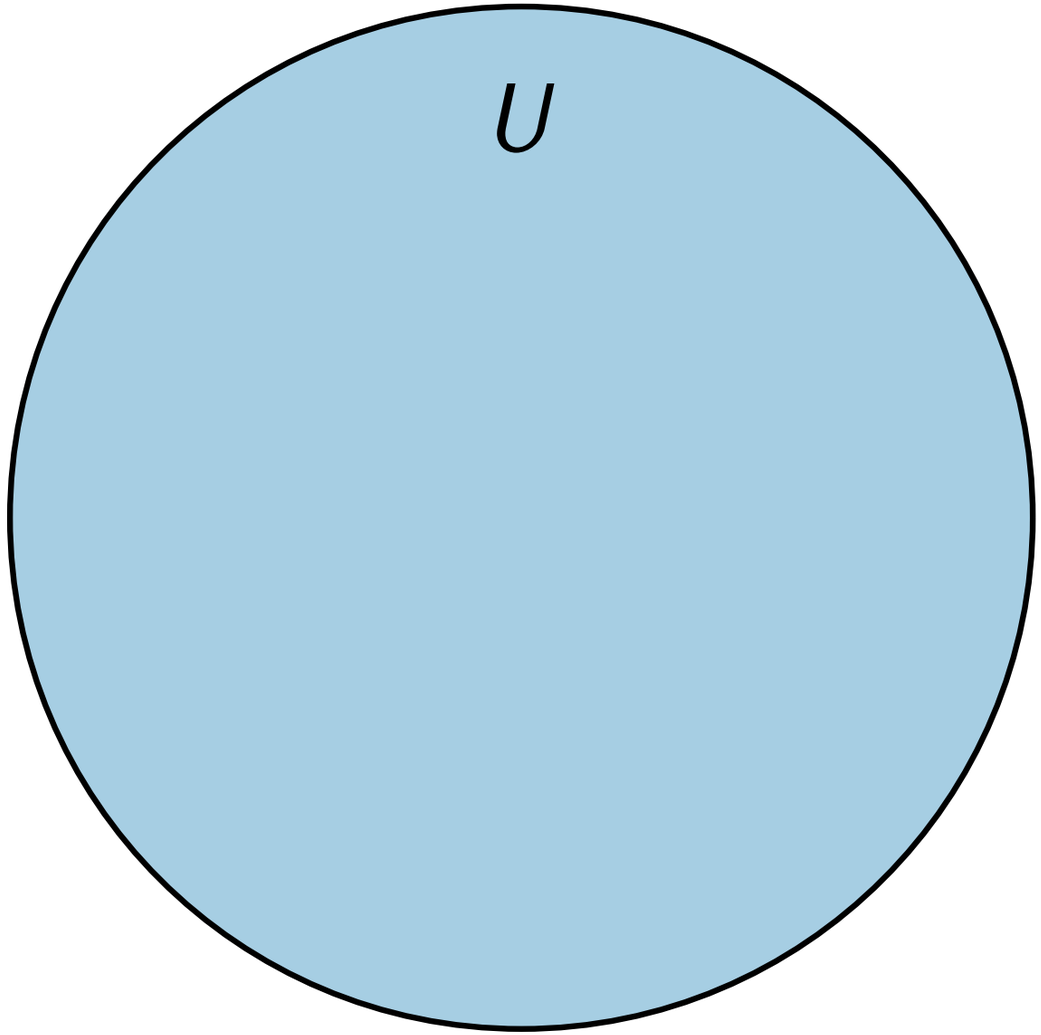


Laufzeiten?
Insert, Delete, Search $\mathcal{O}(1)$ im *schlechtesten Fall*

Aber: Speicherplatz $\sim |U|$

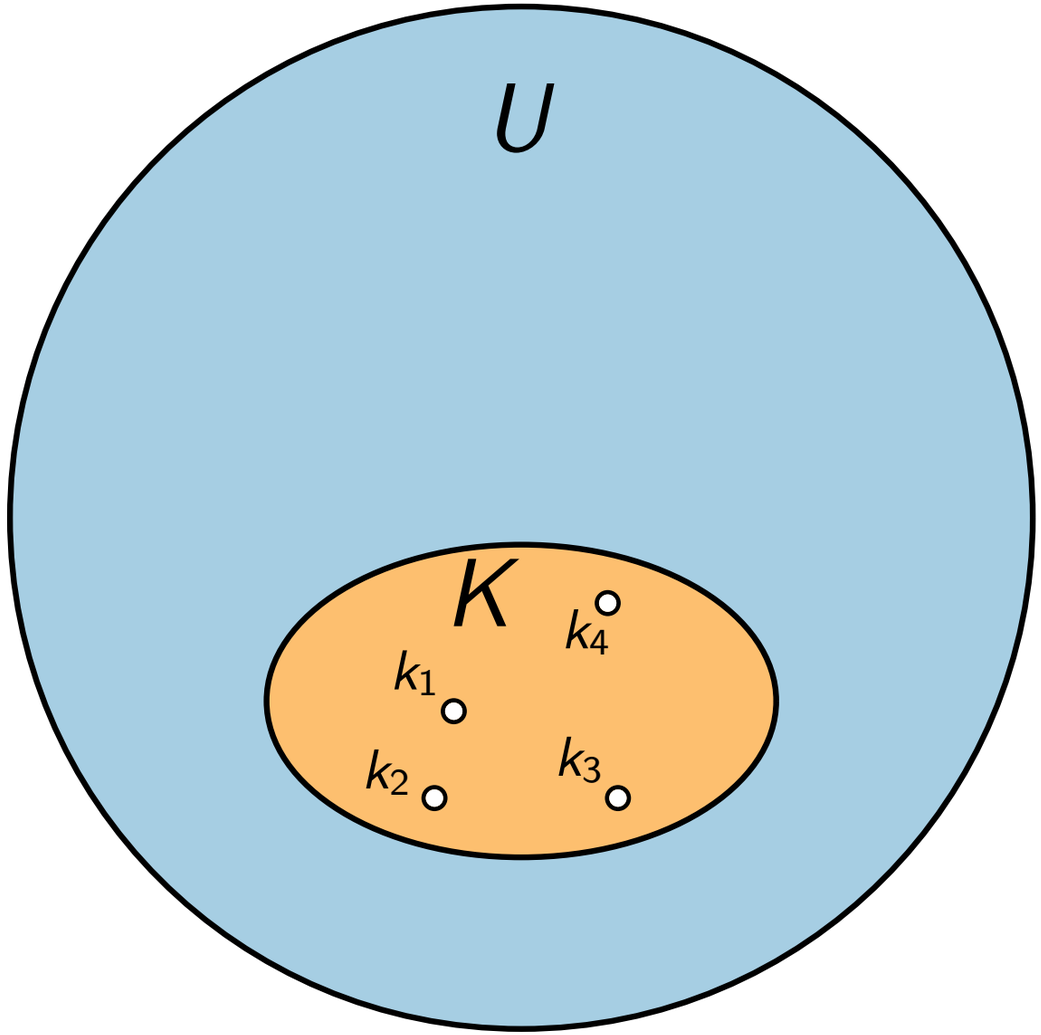
Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$



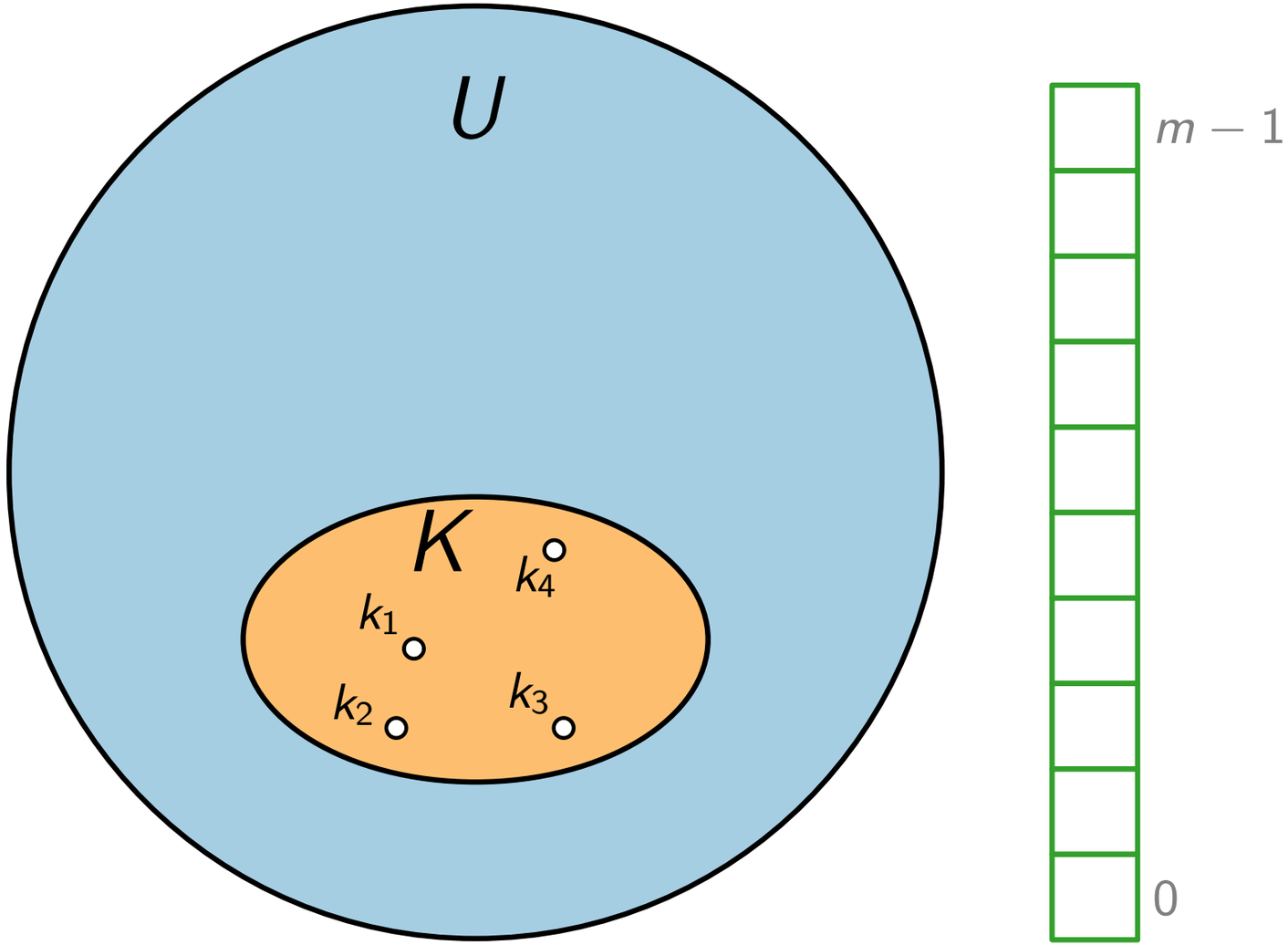
Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$



Hashing mit Verkettung

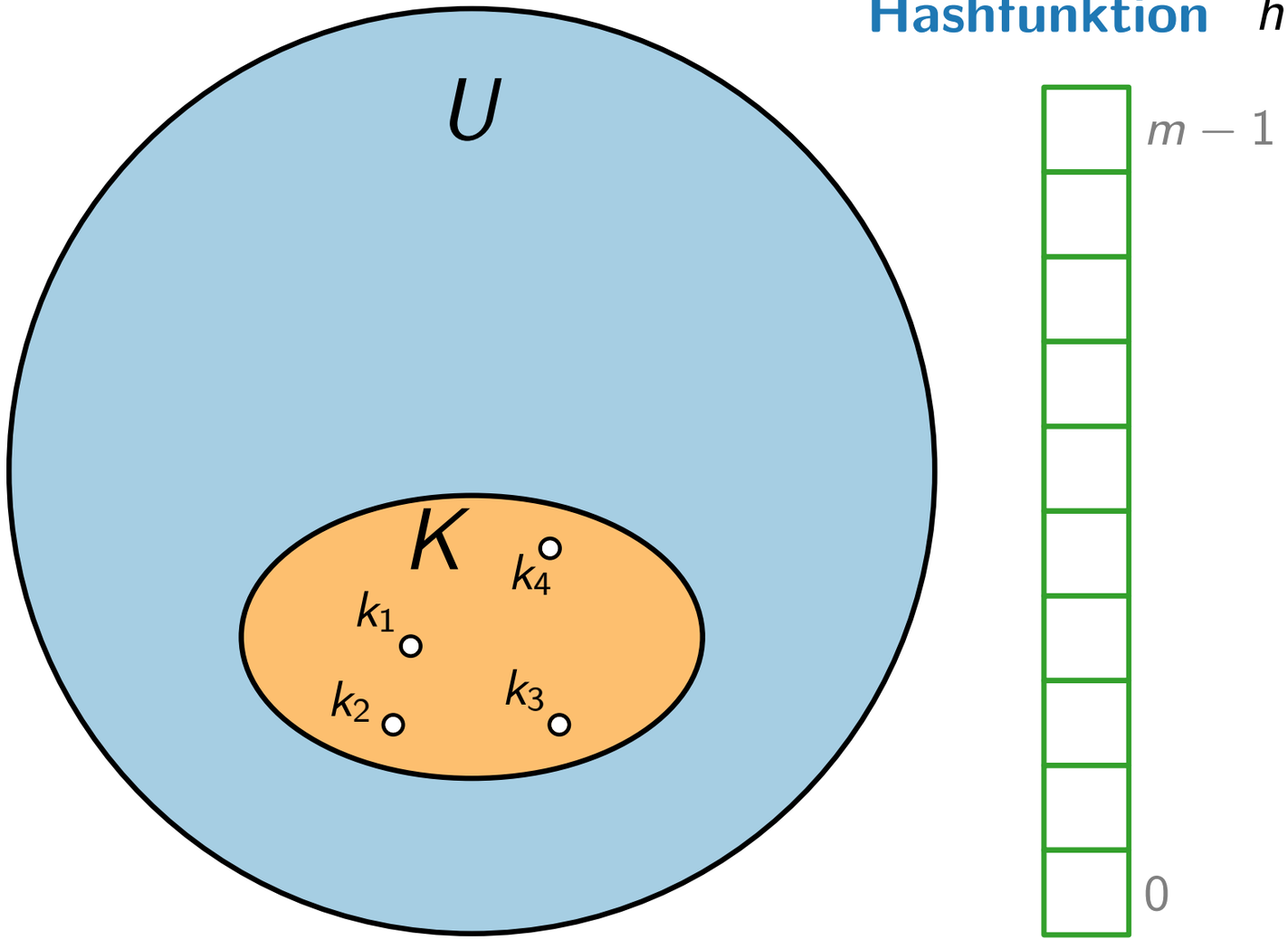
Annahme. **großes** Universum U, d.h. $|U| \gg |K|$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

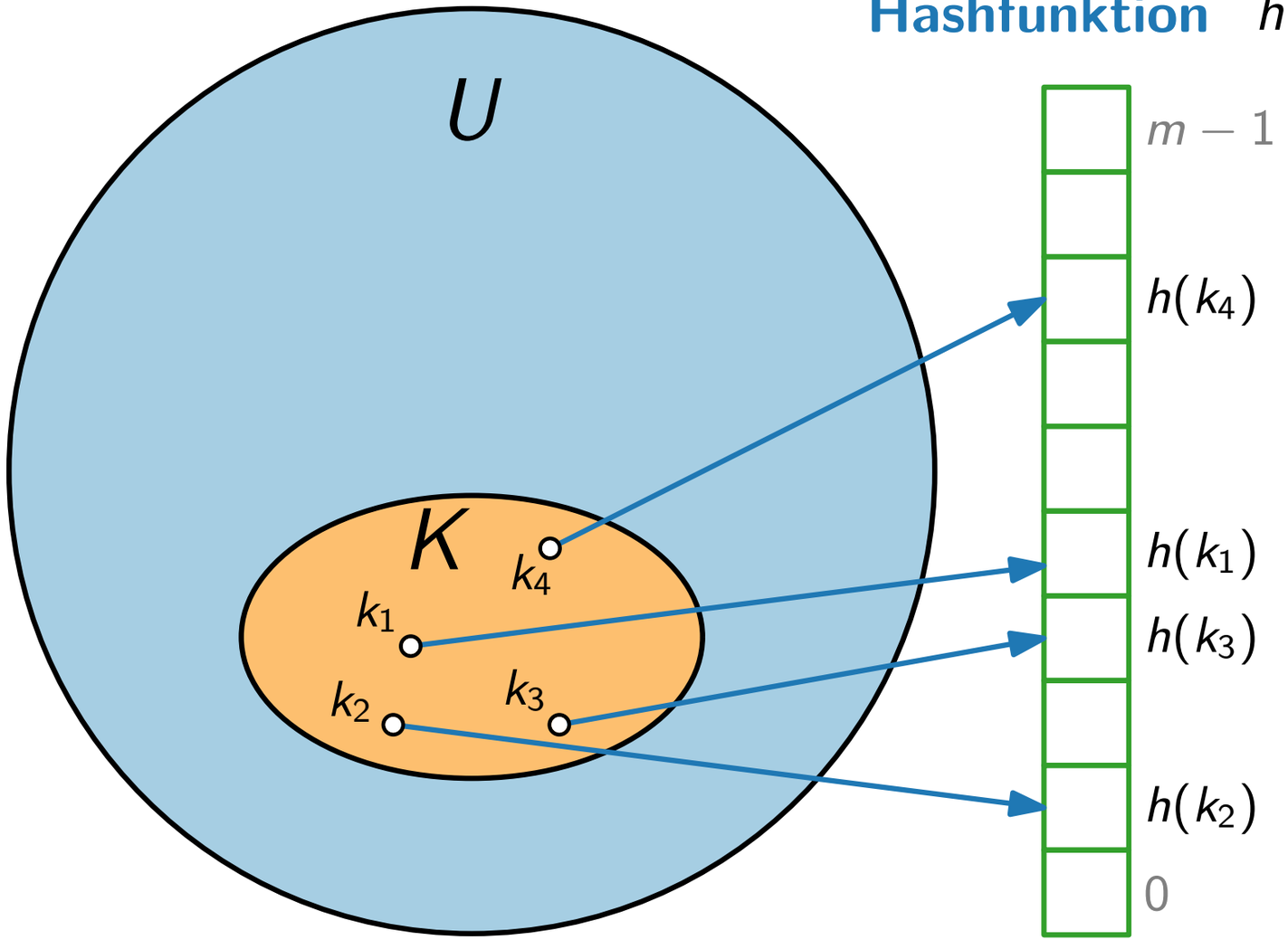
Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

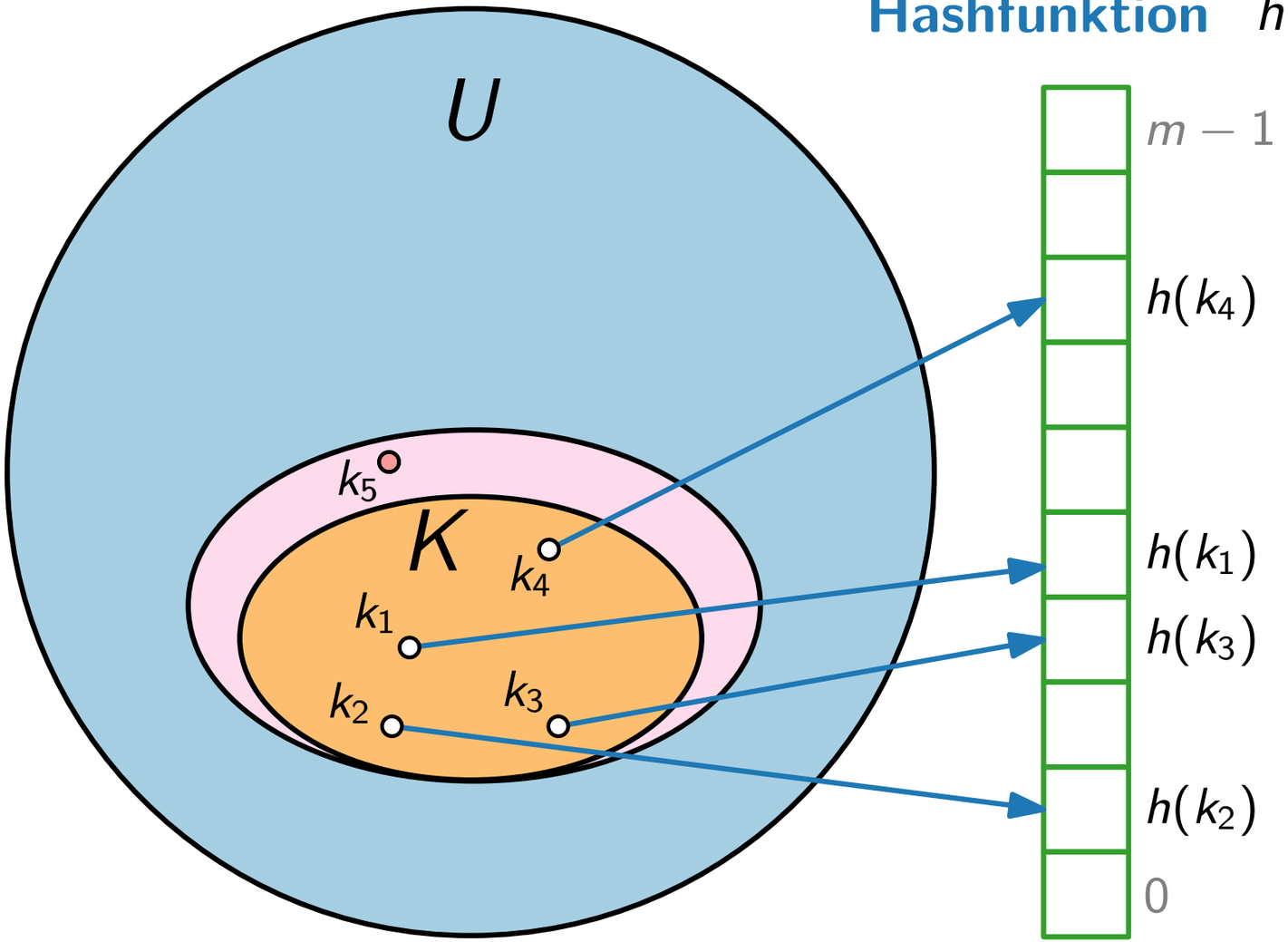
Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

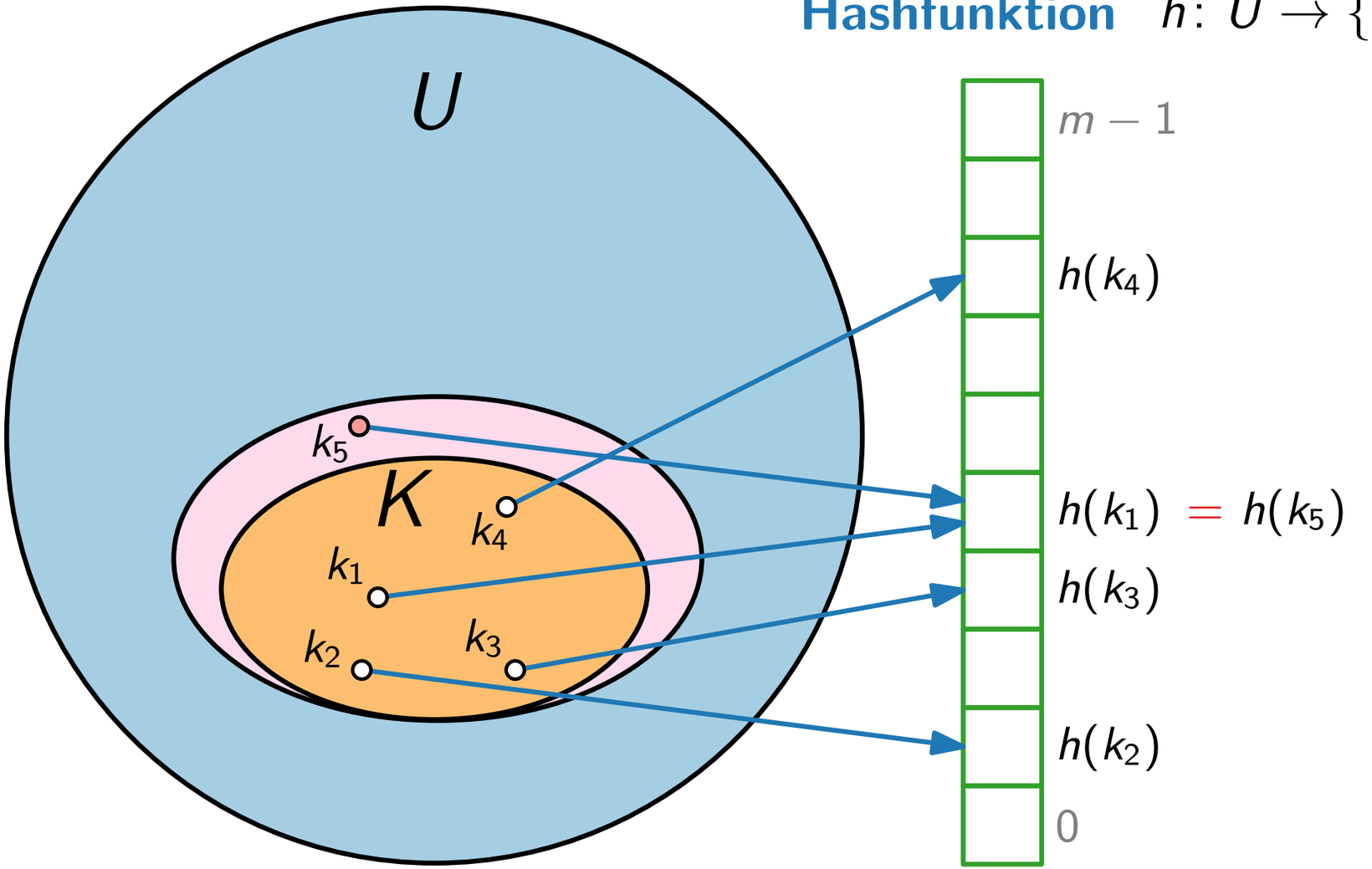
Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

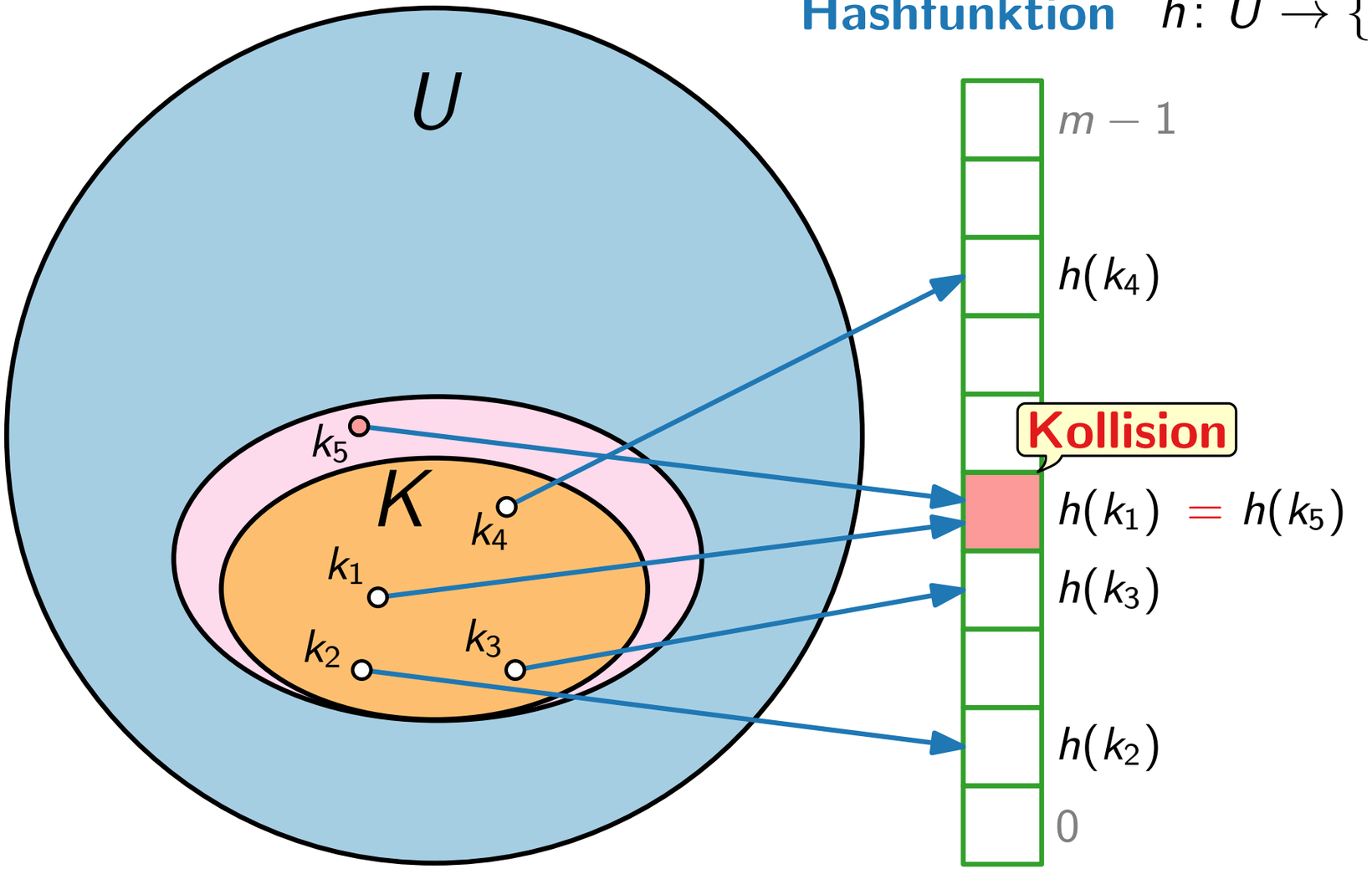
Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

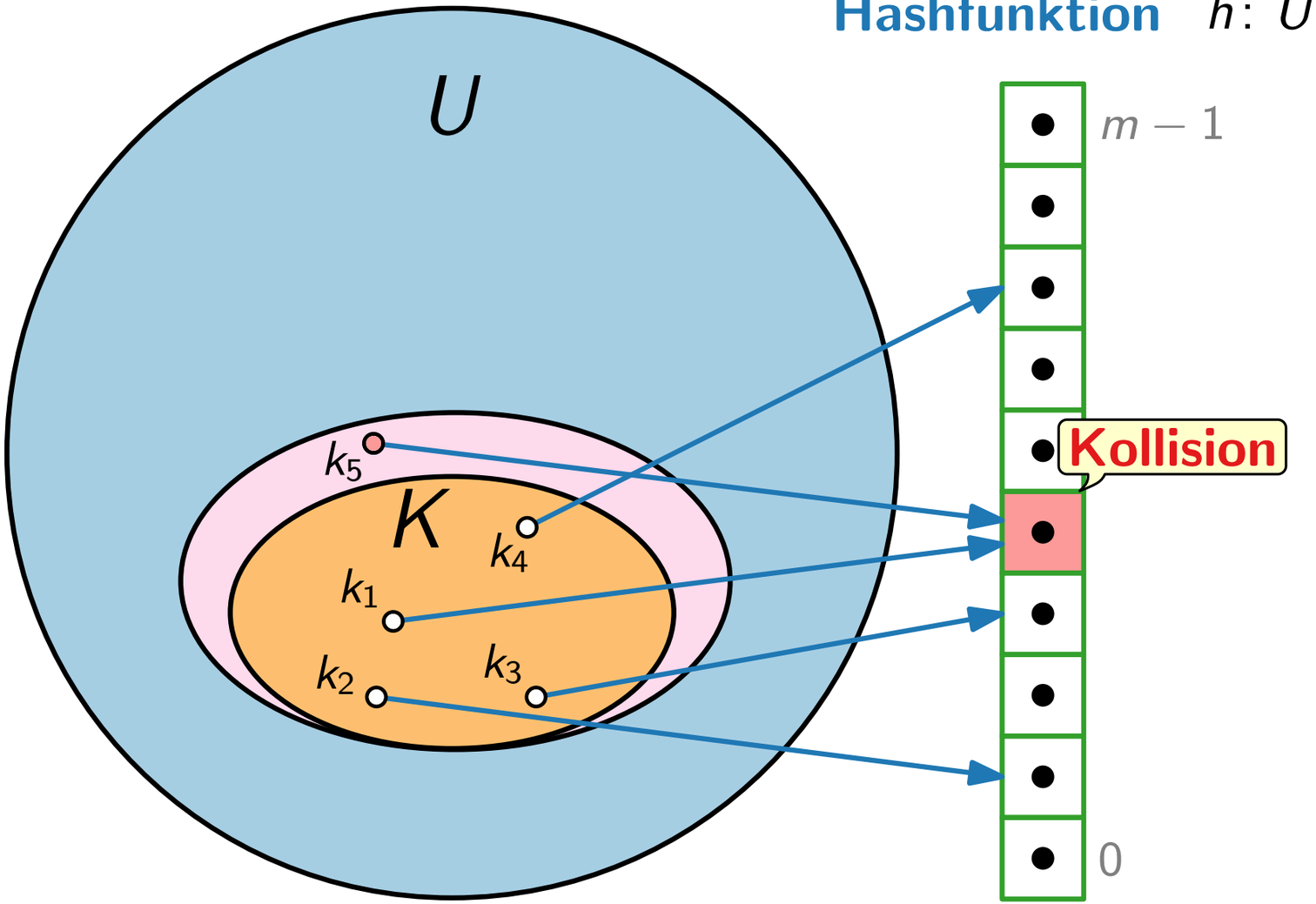
Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

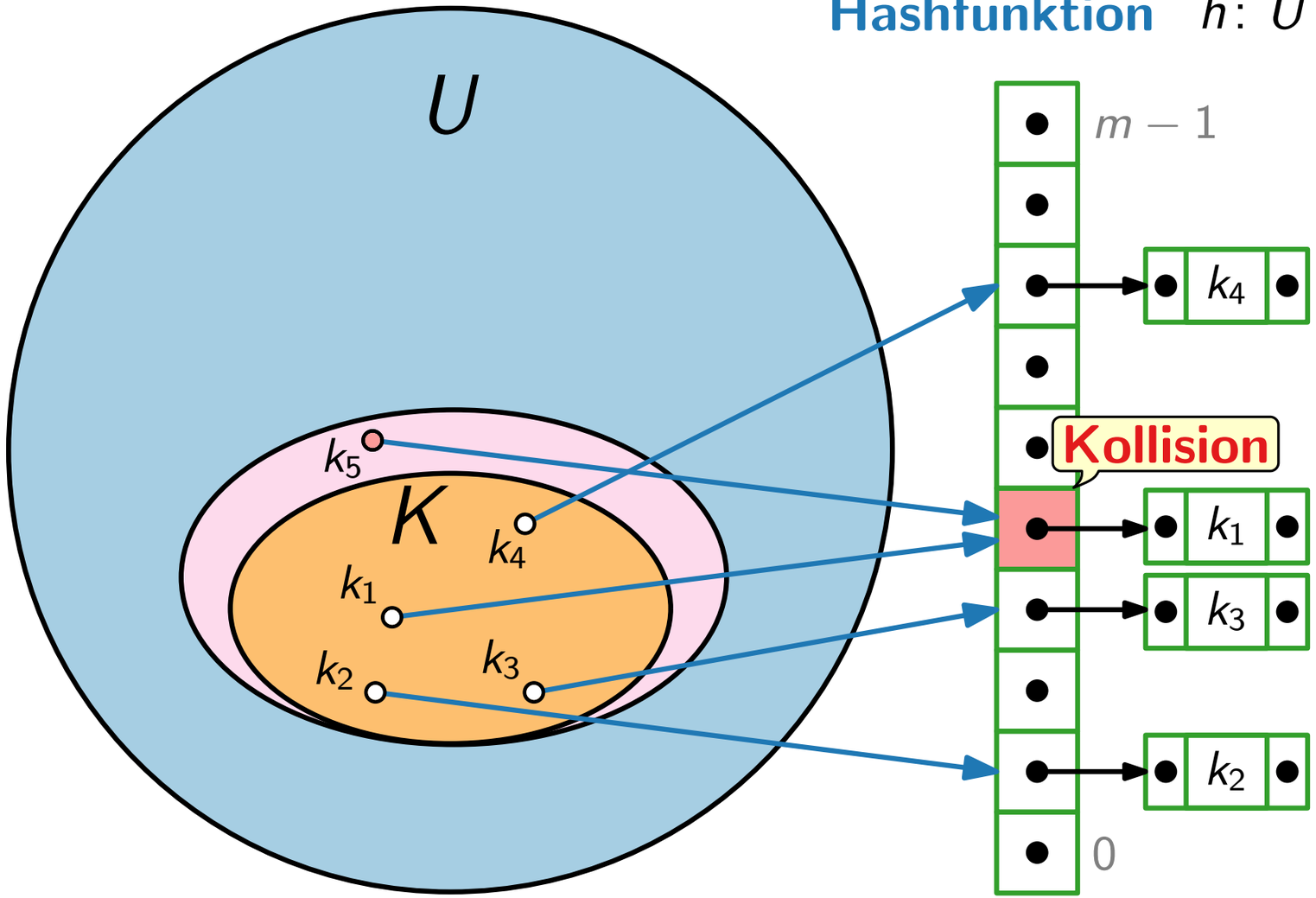
Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

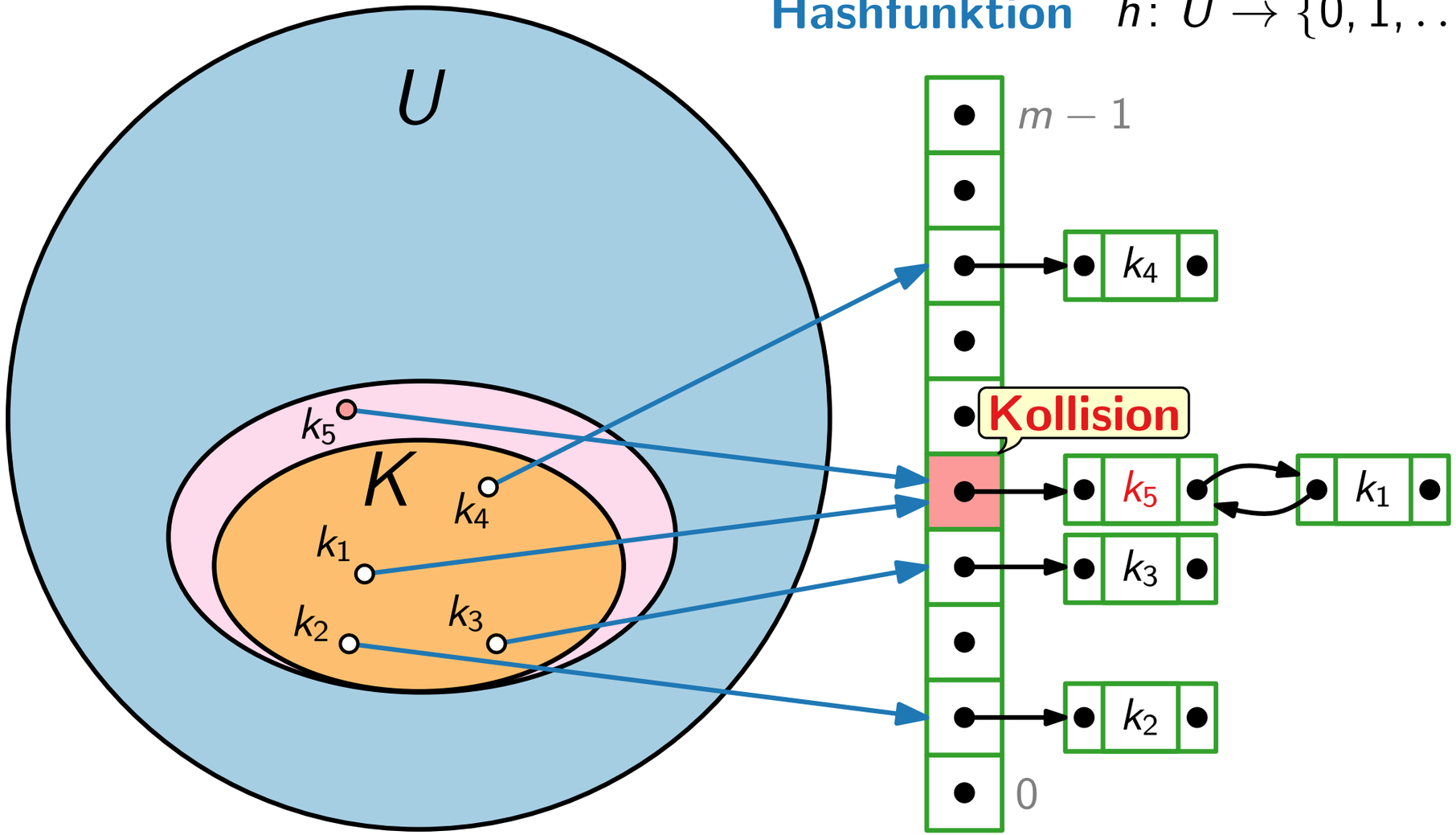
Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Annahme. **großes** Universum U, d.h. $|U| \gg |K|$

Hashfunktion $h\colon U \to \{0, 1, \ldots, m-1\}$



Hashing mit Verkettung

Voraussetzungen:

- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

Operation	**Implementierung**

Hashing mit Verkettung

Voraussetzungen:

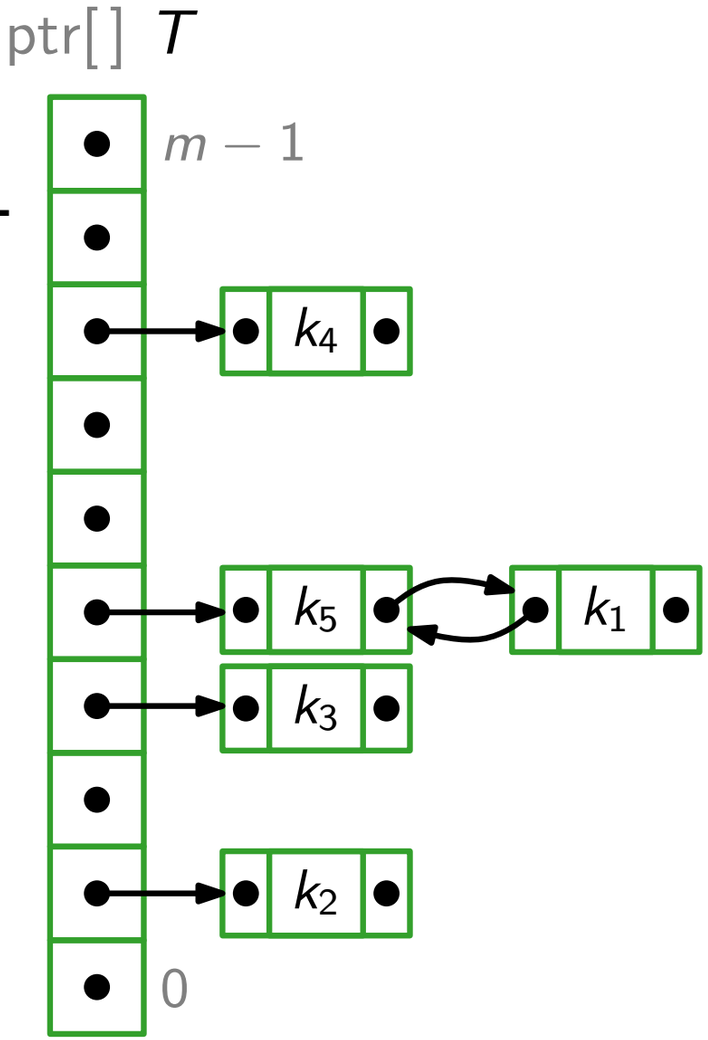
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

Operation	Implementierung
HashChaining(int m)	
ptr Insert(key k)	
Delete(ptr x)	
ptr Search(key k)	

Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

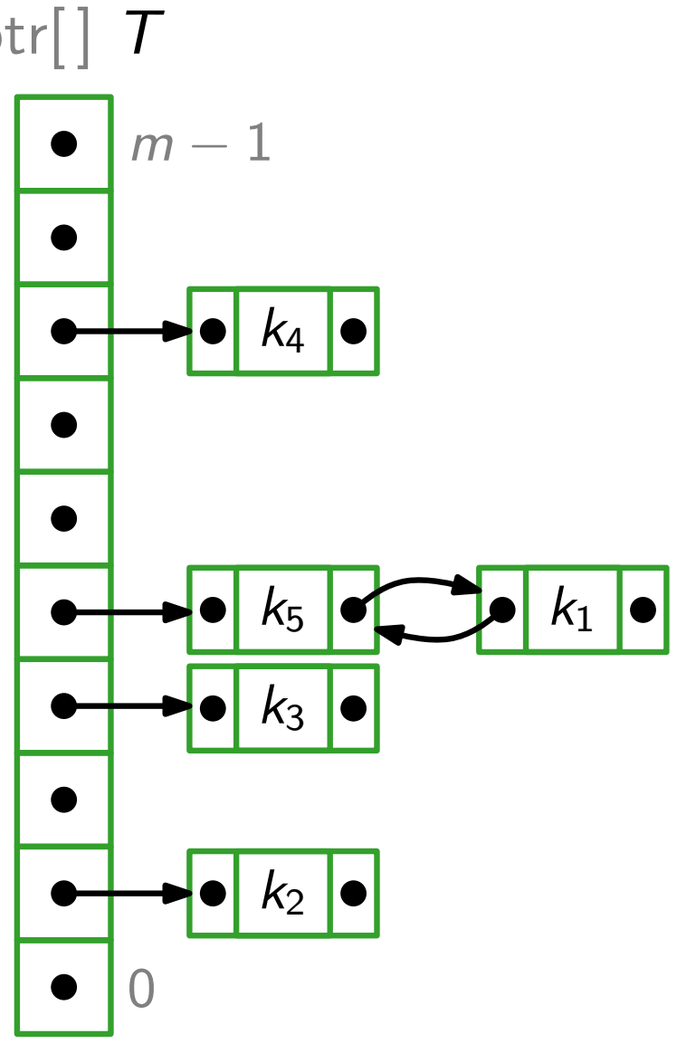
Operation	Implementierung
HashChaining(int m)	
ptr Insert(key k)	
Delete(ptr x)	
ptr Search(key k)	



Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

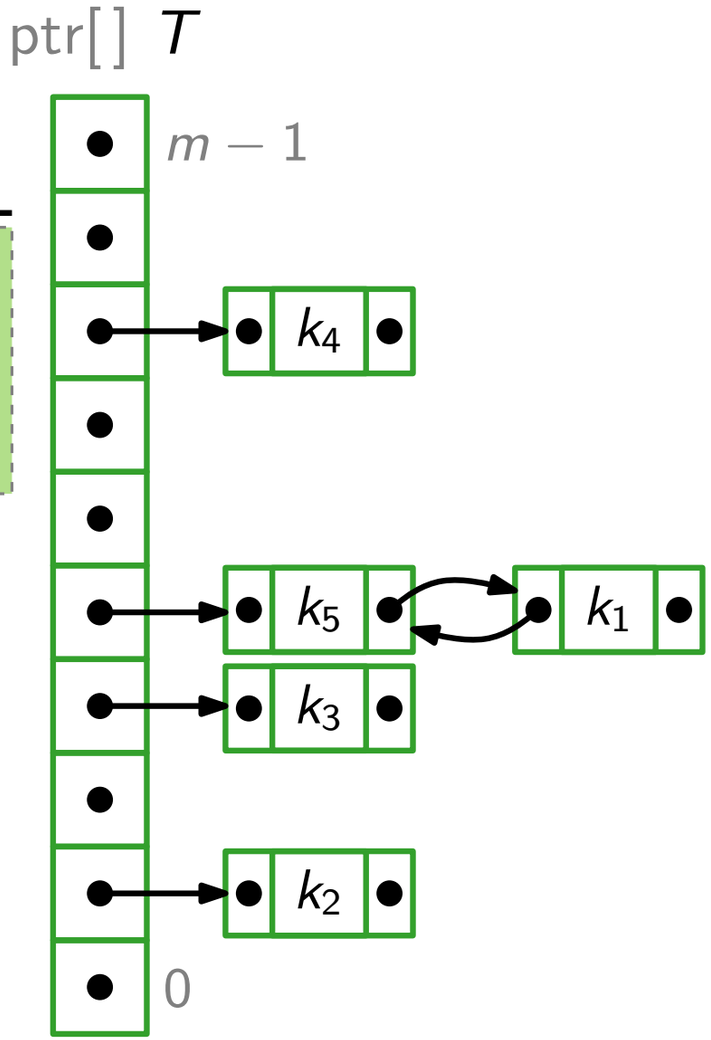
Operation	Implementierung
HashChaining(int m)	$T =$ **new** List$[0 \ldots m-1]$
ptr Insert(key k)	
Delete(ptr x)	
ptr Search(key k)	



Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

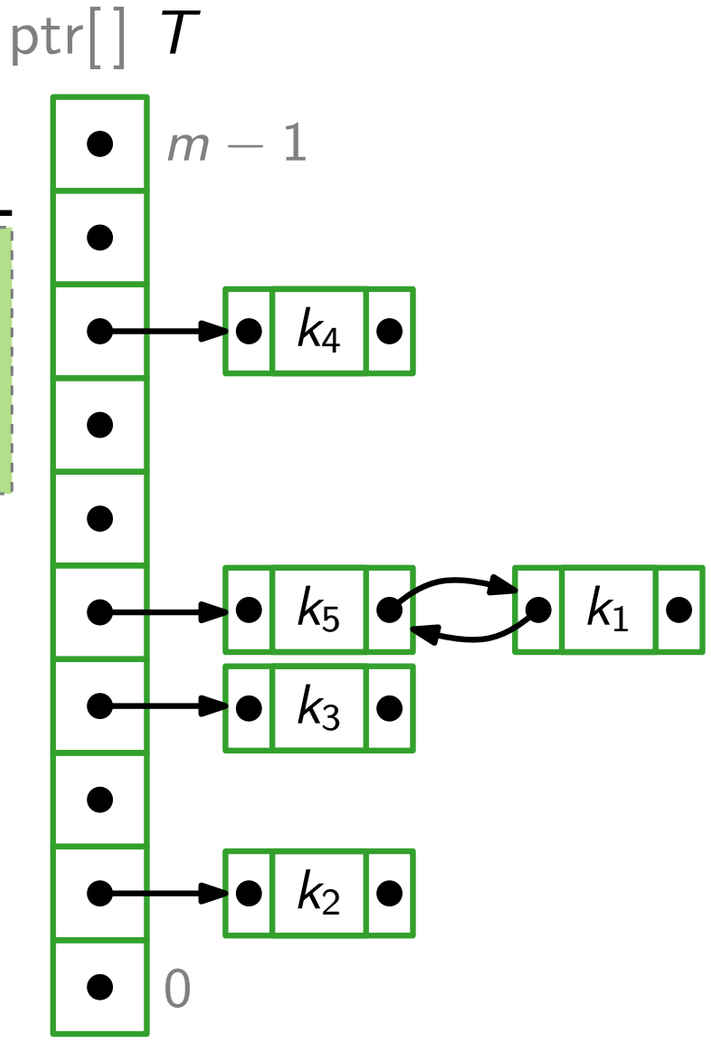
Operation	Implementierung
HashChaining(int m)	T = **new** List$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $\quad T[j]$ = **new** List()
ptr Insert(key k)	
Delete(ptr x)	
ptr Search(key k)	



Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

Operation	Implementierung
HashChaining(int m)	T = **new** List$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $\quad T[j]$ = **new** List()
ptr Insert(key k)	
Delete(ptr x)	
ptr Search(key k)	

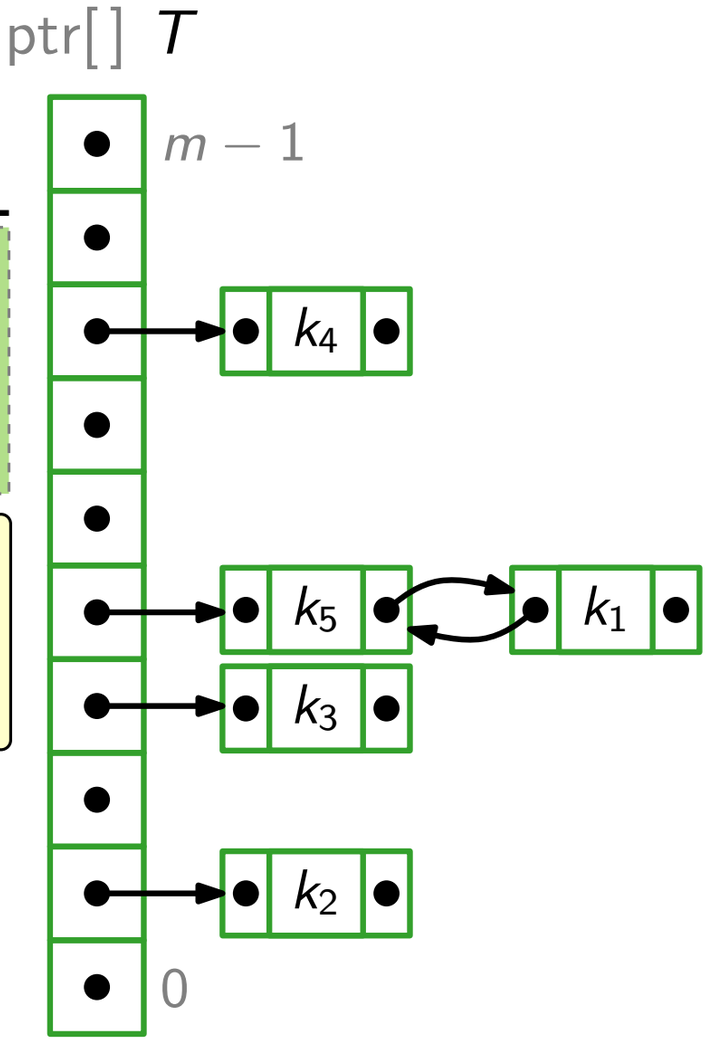


Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

Operation	Implementierung
HashChaining(int m)	T = **new** List$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $\quad T[j]$ = **new** List()
ptr Insert(key k)	
Delete(ptr x)	
ptr Search(key k)	

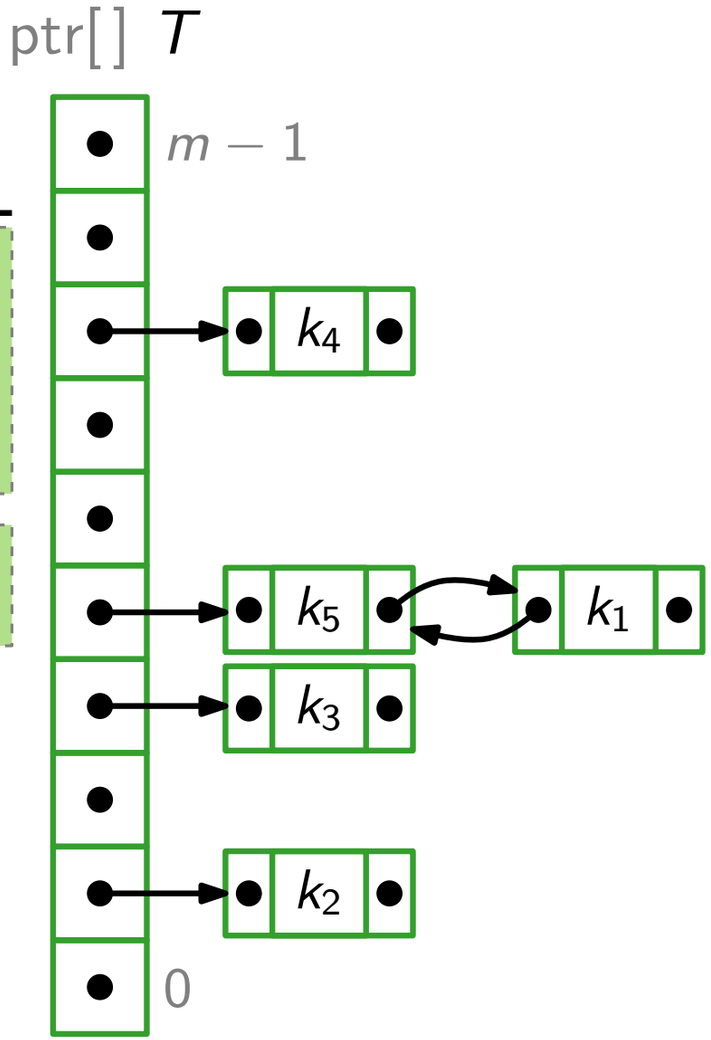
Aufgabe.
Schreiben Sie Insert, Delete & Search.
Verwenden Sie Methoden der DS List!



Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

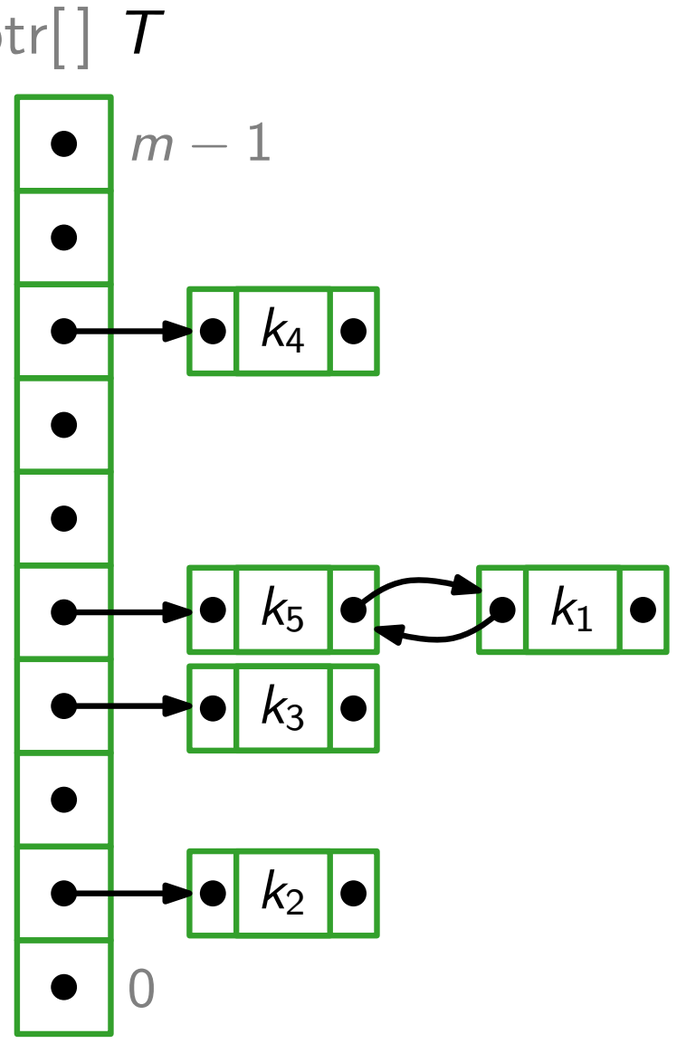
Operation	Implementierung
HashChaining(int m)	T = **new** List$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $\quad T[j]$ = **new** List()
ptr Insert(key k)	**return** $T[h(k)]$.Insert(k)
Delete(ptr x)	
ptr Search(key k)	



Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

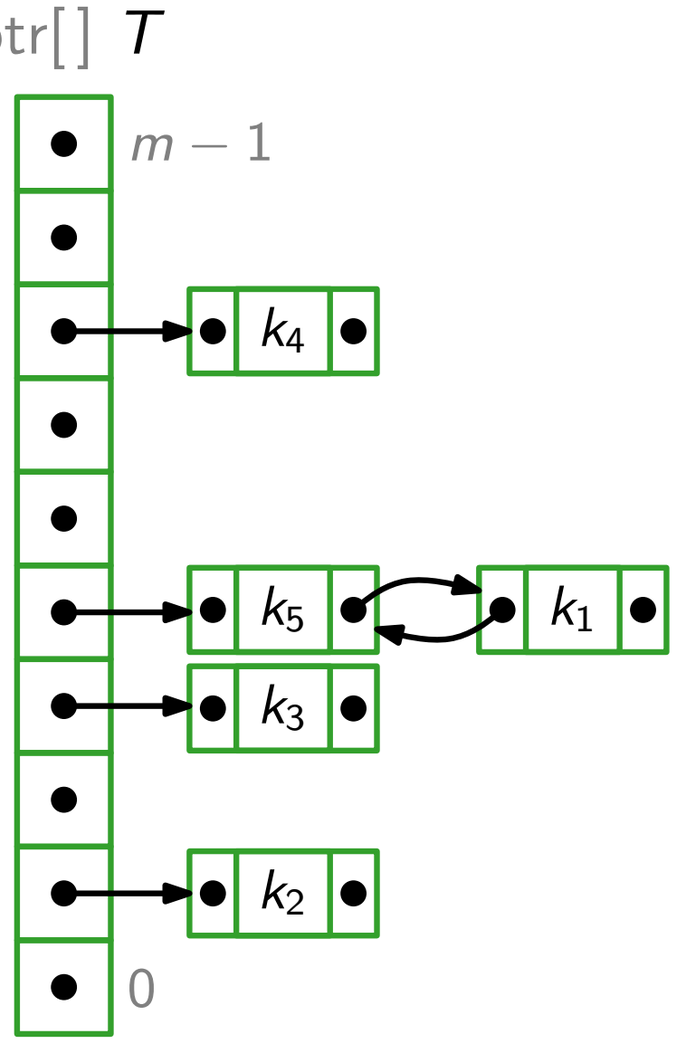
Operation	Implementierung
HashChaining(int m)	T = **new** List$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $\quad T[j]$ = **new** List()
ptr Insert(key k)	**return** $T[h(k)]$.Insert(k)
Delete(ptr x)	$T[h(x.key)]$.Delete(x)
ptr Search(key k)	



Hashing mit Verkettung

Voraussetzungen:
- $|U| \gg |K|$
- Zugriff auf Hashfunktion h

Operation	Implementierung
HashChaining(int m)	T = **new** List$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j]$ = **new** List()
ptr Insert(key k)	**return** $T[h(k)]$.Insert(k)
Delete(ptr x)	$T[h(x.key)]$.Delete(x)
ptr Search(key k)	**return** $T[h(k)]$.Search(k)



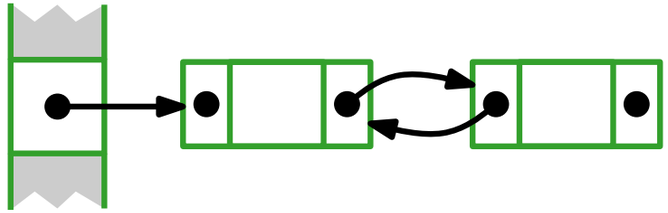
Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

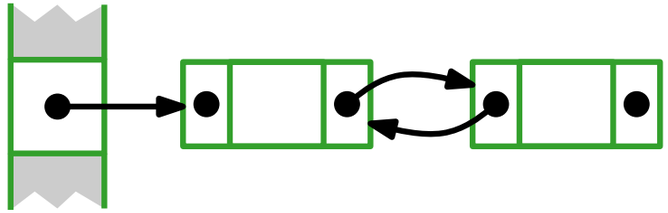
Anmerkung. α ist die durchschnittliche Länge einer **Kette**.



Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

Anmerkung. α ist die durchschnittliche Länge einer **Kette**.

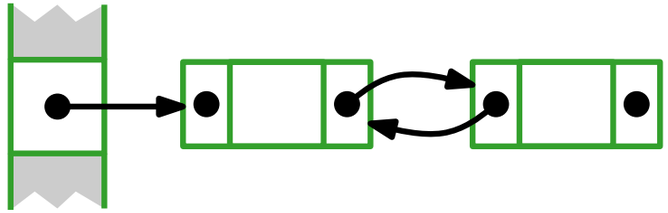


Laufzeit:

Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

Anmerkung. α ist die durchschnittliche Länge einer **Kette**.

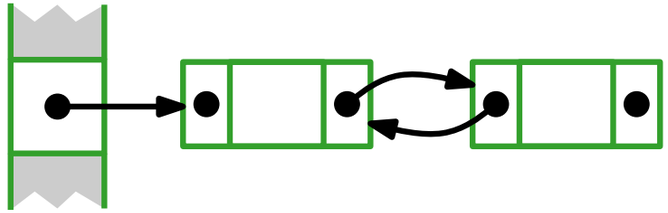


Laufzeit: $\Theta(n)$ im schlimmsten Fall.

Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

Anmerkung. α ist die durchschnittliche Länge einer **Kette**.

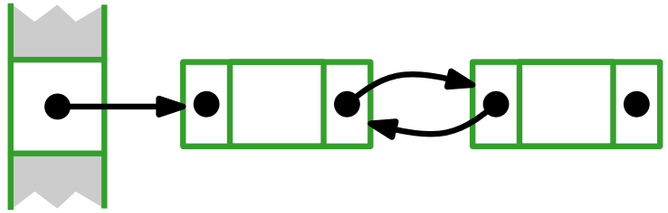


Laufzeit: $\Theta(n)$ im schlimmsten Fall. z.B. $h(k) = 0 \quad \forall k \in K$.

Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

Anmerkung. α ist die durchschnittliche Länge einer **Kette**.



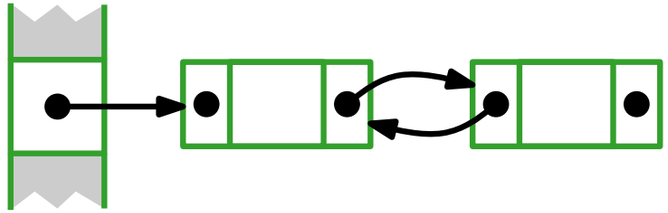
Laufzeit: $\Theta(n)$ im schlimmsten Fall. z.B. $h(k) = 0 \quad \forall k \in K$.

Annahme: **Einfaches uniformes Hashing:**

Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

Anmerkung. α ist die durchschnittliche Länge einer **Kette**.



Laufzeit: $\Theta(n)$ im schlimmsten Fall. z.B. $h(k) = 0 \quad \forall k \in K$.

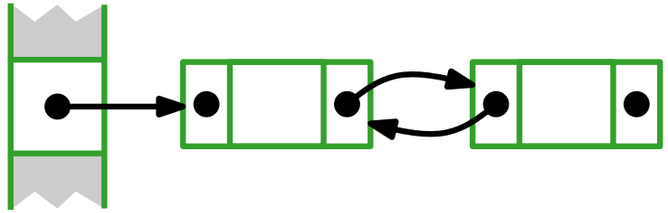
Annahme: **Einfaches uniformes Hashing:**

jedes Element von U wird mit gleicher Wahrscheinlichkeit in jeden der m Einträge der Tabelle gehasht – unabhängig von anderen Elementen.

Analyse

Definition. Die **Auslastung** α einer Hashtabelle sei n/m, also der Quotient der Anzahl der gespeicherten Elemente und der Tabellengröße.

Anmerkung. α ist die durchschnittliche Länge einer **Kette**.



Laufzeit: $\Theta(n)$ im schlimmsten Fall. z.B. $h(k) = 0 \quad \forall k \in K$.

Annahme: **Einfaches uniformes Hashing:**

jedes Element von U wird mit gleicher Wahrscheinlichkeit in jeden der m Einträge der Tabelle gehasht – unabhängig von anderen Elementen.

$$\text{d.h. } \mathbf{Pr}[h(k) = i] = 1/m$$

Suche

Fälle:

1) erfolglose Suche
2) erfolgreiche Suche

Suche

Fälle:

1) erfolglose Suche
2) erfolgreiche Suche

Notation: $n_j = T[j].length$

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$\mathbf{E}[n_j] =$

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$\mathbf{E}[n_j] = n/m$

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolglose** Suche erwartet α Elemente.

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolglose** Suche erwartet α Elemente.

Beweis.

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolglose** Suche erwartet α Elemente.

Beweis. Wenn die Suche nach einem Schlüssel k erfolglos ist, muss $T[h(k)]$ komplett durchsucht werden.

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolglose** Suche erwartet α Elemente.

Beweis. Wenn die Suche nach einem Schlüssel k erfolglos ist, muss $T[h(k)]$ komplett durchsucht werden.

$$\mathbf{E}\big[T[h(k)].length\big] =$$

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolglose** Suche erwartet α Elemente.

Beweis. Wenn die Suche nach einem Schlüssel k erfolglos ist, muss $T[h(k)]$ komplett durchsucht werden.

$$\mathbf{E}\big[T[h(k)].length\big] = \mathbf{E}[n_{h(k)}] =$$

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolglose** Suche erwartet α Elemente.

Beweis. Wenn die Suche nach einem Schlüssel k erfolglos ist, muss $T[h(k)]$ komplett durchsucht werden.

$$\mathbf{E}\big[T[h(k)].length\big] = \mathbf{E}[n_{h(k)}] =$$

Suche

Fälle: 1) erfolglose Suche

2) erfolgreiche Suche

Notation: $n_j = T[j].length$ für $j = 0, 1, \ldots, m-1$.

Dann gilt: $n = n_0 + n_1 + \cdots + n_{m-1}$.

$$\mathbf{E}[n_j] = n/m = \alpha$$

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolglose** Suche erwartet α Elemente.

Beweis. Wenn die Suche nach einem Schlüssel k erfolglos ist, muss $T[h(k)]$ komplett durchsucht werden.

$$\mathbf{E}\big[T[h(k)].length\big] = \mathbf{E}[n_{h(k)}] = \alpha.$$

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens ? Elemente.

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$\#$ durchsuchte Elemente $= \#$ Elemente **vor** x in $T[h(x)]$

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor } x \text{ in } T[h(x)] + 1$$

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor } x \text{ in } T[h(x)] + 1$$

$$= \#\text{ Elemente, die } \textbf{nach } x \text{ in } T[h(x)] \text{ eingefügt wurden} + 1$$

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor } x \text{ in } T[h(x)] + 1$$

$$= \#\text{ Elemente, die } \textbf{nach } x \text{ in } T[h(x)] \text{ eingefügt wurden} + 1$$

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor}\ x \text{ in } T[h(x)] + 1$$

(räumlich)

$$= \#\text{ Elemente, die } \textbf{nach}\ x \text{ in } T[h(x)] \text{ eingefügt wurden} + 1$$

(zeitlich)

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor}\ x \text{ in } T[h(x)] + 1$$

(räumlich)

$$X = \#\text{ Elemente, die } \textbf{nach}\ x \text{ in } T[h(x)] \text{ eingefügt wurden} + 1$$

(zeitlich)

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor}\ x \text{ in } T[h(x)] \quad +1$$

(räumlich)

$$X = \#\text{ Elemente, die } \textbf{nach}\ x \text{ in } T[h(x)] \text{ eingefügt wurden} \quad +1$$

(zeitlich)

Sei $x_1, x_2, \ldots, x_n$ die Folge der Schlüssel in der Reihenfolge des Einfügens.

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor}^{\text{räumlich}}\ x \text{ in } T[h(x)] + 1$$

$$X = \#\text{ Elemente, die } \textbf{nach}^{\text{zeitlich}}\ x \text{ in } T[h(x)] \text{ eingefügt wurden} + 1$$

Sei $x_1, x_2, \ldots, x_n$ die Folge der Schlüssel in der Reihenfolge des Einfügens.

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor}^{\text{räumlich}} x \text{ in } T[h(x)] + 1$$

$$X = \#\text{ Elemente, die } \textbf{nach}^{\text{zeitlich}} x \text{ in } T[h(x)] \text{ eingefügt wurden} + 1$$

Sei $x_1, x_2, \ldots, x_n$ die Folge der Schlüssel in der Reihenfolge des Einfügens.

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] =$

Erfolgreiche Suche

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht eine **erfolgreiche** Suche erwartet höchstens $1 + \alpha/2$ Elemente.

Beweis. Noch eine Annahme:

Jedes der n Elemente in T ist mit gleicher Wahrscheinlichkeit das gesuchte Element x.

$$\#\text{ durchsuchte Elemente} = \#\text{ Elemente } \textbf{vor}^{\text{räumlich}}\ x \text{ in } T[h(x)] + 1$$

$$X = \#\text{ Elemente, die } \textbf{nach}^{\text{zeitlich}}\ x \text{ in } T[h(x)] \text{ eingefügt wurden} + 1$$

Sei $x_1, x_2, \ldots, x_n$ die Folge der Schlüssel in der Reihenfolge des Einfügens.

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$ (einfaches uniformes Hashing!)

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden $+ 1$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden $+ 1$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden $+ 1$

$\mathbf{E}[X] =$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden $+ 1$

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\,[\#\text{ Elemente} \qquad \cdots \qquad]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)\right]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\Big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}_{\|}\Big]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\,[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}_{\|}]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\Big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}_{\sum_{j=i+1}^{n} X_{ij}}\Big]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}_{\stackrel{n}{\sum_{j=i+1}} X_{ij}}\big]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\,[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}_{\overset{n}{\sum_{j=i+1}} X_{ij}}]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}]$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}_{\mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right]}\big]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}]$$

$\mathbf{E}[X_{ij}] = 1/m$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n}\mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n}\mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\mathbf{E}\left[\sum_{j=i+1}^{n}{}^{\|} X_{ij}\right]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\mathbf{E}[X_{ij}] = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1$$

($\mathbf{E}[X_{ij}] = 1/m$)

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1$$

$\mathbf{E}[X_{ij}]$: $1/m$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}\big]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \quad (\|\ n-i)$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1$$

($\mathbf{E}[X_{ij}]$: $1/m$)

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \overset{\|}{n-i}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \underbrace{\mathbf{E}[X_{ij}]}_{1/m} = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}\big]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \quad (\|: n-i)$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad (\text{Setze } k = n - i)$$

Definiere Indikator-Zufallsvariable: $X_{ij} = 1$ falls $h(x_i) = h(x_j)$.

Klar: $\mathbf{E}[X_{ij}] = \mathbf{Pr}[X_{ij} = 1] = 1/m$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \overset{\|}{n-i}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad \text{Setze } k = n - i$$

$$= 1 + \frac{1}{nm}\sum_{k=0}^{n-1} k$$

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}\big]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \quad (\|\ n-i)$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad (\text{Setze } k = n-i)$$

$$= 1 + \frac{1}{nm}\sum_{k=0}^{n-1} k$$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \quad (\|\ n-i)$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 \quad = \quad 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad \text{Setze } k = n-i$$

$$= 1 + \frac{1}{nm}\sum_{k=0}^{n-1} k \quad = 1 + \frac{n(n-1)}{2nm}$$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \quad (\|\ n-i)$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad (\text{Setze } k = n-i)$$

$$= 1 + \frac{1}{nm}\sum_{k=0}^{n-1} k = 1 + \frac{n(n-1)}{2nm} = 1 + \frac{n-1}{2m}$$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \overset{\|}{n-i}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad \text{Setze } k = n - i$$

$$= 1 + \frac{1}{nm}\sum_{k=0}^{n-1} k = 1 + \frac{n(n-1)}{2nm} = 1 + \frac{n-1}{2m} <$$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[\underbrace{\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)}\big]$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \quad (\|\ n-i)$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad \text{Setze } k = n-i$$

$$= 1 + \frac{1}{nm}\sum_{k=0}^{n-1} k = 1 + \frac{n(n-1)}{2nm} = 1 + \frac{n-1}{2m} < \quad (\alpha = n/m)$$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Erfolgreiche Suche

X = # Elemente, die **nach** x in $T[h(x)]$ eingefügt wurden + 1

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\big[1 + \#\text{ Elemente, die nach } x_i \text{ in } T[h(x_i)] \text{ eingefügt wurden}\big]$$

$$= \frac{1}{n}\sum_{i=1}^{n} 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\underbrace{[\#\text{ Elemente } x_j \text{ mit } j > i \text{ und } h(x_i) = h(x_j)]}$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}\left[\sum_{j=i+1}^{n} X_{ij}\right] \quad (\|\ n-i)$$

$$= 1 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}] \quad (1/m) \quad = 1 + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=i+1}^{n} 1 = 1 + \frac{1}{nm}\sum_{i=1}^{n} n - i \quad (\text{Setze } k = n-i)$$

$$= 1 + \frac{1}{nm}\sum_{k=0}^{n-1} k = 1 + \frac{n(n-1)}{2nm} = 1 + \frac{n-1}{2m} < 1 + \frac{\alpha}{2} \quad (\alpha = n/m)$$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

- **erfolgreiche** Suche erwartet höch. $1 + \alpha/2$ Elemente

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

- **erfolgreiche** Suche erwartet höch. $1 + \alpha/2$ Elemente
- **erfolglose** Suche erwartet α Elemente.

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

- **erfolgreiche** Suche erwartet höch. $1 + \alpha/2$ Elemente
- **erfolglose** Suche erwartet α Elemente.

Und **Einfügen**?

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

- **erfolgreiche** Suche erwartet höch. $1 + \alpha/2$ Elemente
- **erfolglose** Suche erwartet α Elemente.

Und **Einfügen**? Und **Löschen**?

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

- **erfolgreiche** Suche erwartet höch. $1 + \alpha/2$ Elemente
- **erfolglose** Suche erwartet α Elemente.

Und **Einfügen**? Und **Löschen**?

Satz. Unter der Annahme des einfachen uniformen Hashings laufen **alle** Wörterbuch-Operationen in (erwartet) konstanter Zeit, falls ... ?

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

- **erfolgreiche** Suche erwartet höch. $1 + \alpha/2$ Elemente
- **erfolglose** Suche erwartet α Elemente.

Und **Einfügen**? Und **Löschen**?

Satz. Unter der Annahme des einfachen uniformen Hashings laufen **alle** Wörterbuch-Operationen in (erwartet) konstanter Zeit, falls $n \in \mathcal{O}(m)$.

Zusammenfassung Ergebnisse

Satz. Unter der Annahme des einfachen uniformen Hashings durchsucht beim Hashing mit Verkettung eine

- **erfolgreiche** Suche erwartet höch. $1 + \alpha/2$ Elemente
- **erfolglose** Suche erwartet α Elemente.

Und **Einfügen**? Und **Löschen**?

Satz. Unter der Annahme des einfachen uniformen Hashings laufen **alle** Wörterbuch-Operationen in (erwartet) konstanter Zeit, falls $n \in \mathcal{O}(m)$.

$\Rightarrow \frac{n}{m}$ konstant

Was ist eine gute Hashfunktion?

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.
   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: $U =$ Zeichenketten, $K =$ Wörter der deutschen Sprache

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: $U =$ Zeichenketten, $K =$ Wörter der deutschen Sprache

     h : nimm die ersten drei Buchstaben $\rightarrow$ Zahl

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: $U =$ Zeichenketten, $K =$ Wörter der deutschen Sprache

     h : nimm die ersten drei Buchstaben $\rightarrow$ Zahl

     schlecht:

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: $U =$ Zeichenketten, $K =$ Wörter der deutschen Sprache

     h : nimm die ersten drei Buchstaben $\rightarrow$ Zahl

     schlecht:
     - viele Wörter fangen mit „sch“ an

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: $U =$ Zeichenketten, $K =$ Wörter der deutschen Sprache

     h : nimm die ersten drei Buchstaben $\rightarrow$ Zahl

     schlecht:
     - viele Wörter fangen mit „sch“ an
       $\Rightarrow$ selber Hashwert

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: U = Zeichenketten, K = Wörter der deutschen Sprache

     h : nimm die ersten drei Buchstaben $\rightarrow$ Zahl

     schlecht:
     - viele Wörter fangen mit „sch“ an
       $\Rightarrow$ selber Hashwert
     - andere Buchstaben haben keinen Einfluss

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: U = Zeichenketten, K = Wörter der deutschen Sprache

     h : nimm die ersten drei Buchstaben $\rightarrow$ Zahl

     schlecht:
     - viele Wörter fangen mit „sch“ an
       $\Rightarrow$ selber Hashwert
     - andere Buchstaben haben keinen Einfluss

2.

Was ist eine gute Hashfunktion?

1. so „zufällig“ wie möglich – um der Annahme des einfachen uniformen Hashings möglichst nahe zu kommen.

   - Hashfunktion sollte die Schlüssel aus dem Universum U möglichst gleichmäßig über die m Plätze der Hashtabelle verteilen.

   - Hashfunktion sollte Muster in der Schlüsselmenge K gut **auflösen**.

     Beispiel: U = Zeichenketten, K = Wörter der deutschen Sprache

     h : nimm die ersten drei Buchstaben $\rightarrow$ Zahl

     schlecht:
     - viele Wörter fangen mit „sch“ an
       $\Rightarrow$ selber Hashwert
     - andere Buchstaben haben keinen Einfluss

2. einfach zu berechnen!

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

American
Standard
Code of
Information
Interchange

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

American
Standard
Code of
Information
Interchange

Zum Beispiel:

AW

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

American
Standard
Code of
Information
Interchange

Zum Beispiel:

AW $\to (65, 87)_{10}$

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

American
Standard
Code of
Information
Interchange

Zum Beispiel:

AW $\to (65, 87)_{10} = (1000001, 1010111)_2$

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

American
Standard
Code of
Information
Interchange

Zum Beispiel:

$$\text{AW} \;\to\; (65, 87)_{10} \;=\; (1000001, 1010111)_2 \;\to\; 1000001\,1010111_2$$

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

American Standard Code of Information Interchange

Zum Beispiel:

$$\text{AW} \;\to\; (65, 87)_{10} \;=\; (1000001, 1010111)_2 \;\to\; 1000001\,1010111_2 \;=\; 65 \cdot 128 + 87$$

Annahme: Alle Schlüssel sind (natürliche) Zahlen.

Suche also Hashfunktionen: $\mathbb{N} \to \{0, \ldots, m-1\}$

Rechtfertigung:

33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		
48	0	64	@	80	P	96	‘	112	p		

American Standard Code of Information Interchange

Zum Beispiel:

$$\text{AW} \;\to\; (65, 87)_{10} \;=\; (1000001, 1010111)_2 \;\to\; 1000001\,1010111_2 \;=\; 65 \cdot 128 + 87 \;=\; 8407_{10}$$

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

$h(1026) = 2$

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

$h(1026) = 2$

$h(2050) = 2$

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

$h(1026) = 2 \qquad 1026_{10} = 0100000000010_2$

$h(2050) = 2 \qquad 2050_{10} = 1000000000010_2$

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

10 niedrigwertigste Stellen

$h(1026) = 2 \qquad 1026_{10} = 0100000000010_2$

$h(2050) = 2 \qquad 2050_{10} = 1000000000010_2$

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

10 niedrigwertigste Stellen

$h(1026) = 2 \qquad 1026_{10} = 0100000000010_2$

$h(2050) = 2 \qquad 2050_{10} = 1000000000010_2$

d.h. die 2 höherwertigsten Stellen werden von h ignoriert

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

10 niedrigwertigste Stellen

$h(1026) = 2 \qquad 1026_{10} = 0100000000010_2$

$h(2050) = 2 \qquad 2050_{10} = 1000000000010_2$

d.h. die 2 höherwertigsten Stellen werden von h ignoriert

Moral: vermeide $m =$ Zweierpotenz

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

10 niedrigwertigste Stellen

$h(1026) = 2 \qquad 1026_{10} = 0100000000010_2$

$h(2050) = 2 \qquad 2050_{10} = 1000000000010_2$

d.h. die 2 höherwertigsten Stellen werden von h ignoriert

Moral: vermeide m = Zweierpotenz

Strategie: wähle für m eine Primzahl, entfernt von Zweierpotenz

Divisionsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto k \bmod m$$

Beispiel: $h(k) = k \bmod 1024$

10 niedrigwertigste Stellen

$h(1026) = 2 \qquad 1026_{10} = 0100000000010_2$

$h(2050) = 2 \qquad 2050_{10} = 1000000000010_2$

d.h. die 2 höherwertigsten Stellen werden von h ignoriert

Moral: vermeide m = Zweierpotenz

Strategie: wähle für m eine Primzahl, entfernt von Zweierpotenz

löst Muster gut auf

Multiplikationsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto \lfloor m \cdot (\, kA \bmod 1\,) \rfloor\,, \quad \text{wobei } 0 < A < 1.$$

Multiplikationsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto \lfloor m \cdot (\underbrace{kA \bmod 1}_{\text{gebrochener Anteil von } kA}) \rfloor \,, \quad \text{wobei } 0 < A < 1.$$

Multiplikationsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto \lfloor m \cdot (\underbrace{kA \bmod 1}_{\text{gebrochener Anteil von } kA \text{ d.h. } kA - \lfloor kA \rfloor}) \rfloor \,, \quad \text{wobei } 0 < A < 1.$$

Multiplikationsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto \lfloor m \cdot (\underbrace{kA \bmod 1}_{\substack{\text{gebrochener Anteil von } kA \\ \text{d.h. } kA - \lfloor kA \rfloor}}) \rfloor \,, \quad \text{wobei } 0 < A < 1.$$

- Verschiedene Werte von A „funktionieren“ verschieden gut.

Multiplikationsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto \lfloor m \cdot (\underbrace{kA \bmod 1}_{\text{gebrochener Anteil von } kA \text{, d.h. } kA - \lfloor kA \rfloor}) \rfloor \text{ , wobei } 0 < A < 1.$$

- Verschiedene Werte von A „funktionieren“ verschieden gut.

  gut: z.B. $A \approx \frac{\sqrt{5}-1}{2}$ [Knuth: The Art of Computer Programming III, '73]

Multiplikationsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto \lfloor m \cdot (\underbrace{kA \bmod 1}_{\text{gebrochener Anteil von } kA \text{, d.h. } kA - \lfloor kA \rfloor}) \rfloor \text{, wobei } 0 < A < 1.$$

- Verschiedene Werte von A „funktionieren“ verschieden gut.

  gut: z.B. $A \approx \frac{\sqrt{5}-1}{2}$ [Knuth: The Art of Computer Programming III, '73]

- Vorteil gegenüber Divisionsmethode: Wahl von m relativ beliebig.

Multiplikationsmethode

Hashfunktion $h\colon \mathbb{N} \to \{0, \ldots, m-1\}$

$$k \mapsto \lfloor m \cdot (\underbrace{kA \bmod 1}_{\text{gebrochener Anteil von } kA \text{, d.h. } kA - \lfloor kA \rfloor}) \rfloor \text{ , wobei } 0 < A < 1.$$

- Verschiedene Werte von A „funktionieren“ verschieden gut.

  gut: z.B. $A \approx \frac{\sqrt{5}-1}{2}$ [Knuth: The Art of Computer Programming III, '73]

- Vorteil gegenüber Divisionsmethode: Wahl von m relativ beliebig.

  Insbesondere $m =$ Zweierpotenz möglich.
  $\Rightarrow$ schnell berechenbar (in Java verschiebt `a << s` die Dualzahlendarstellung von a um s Stellen nach links)

Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen

Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

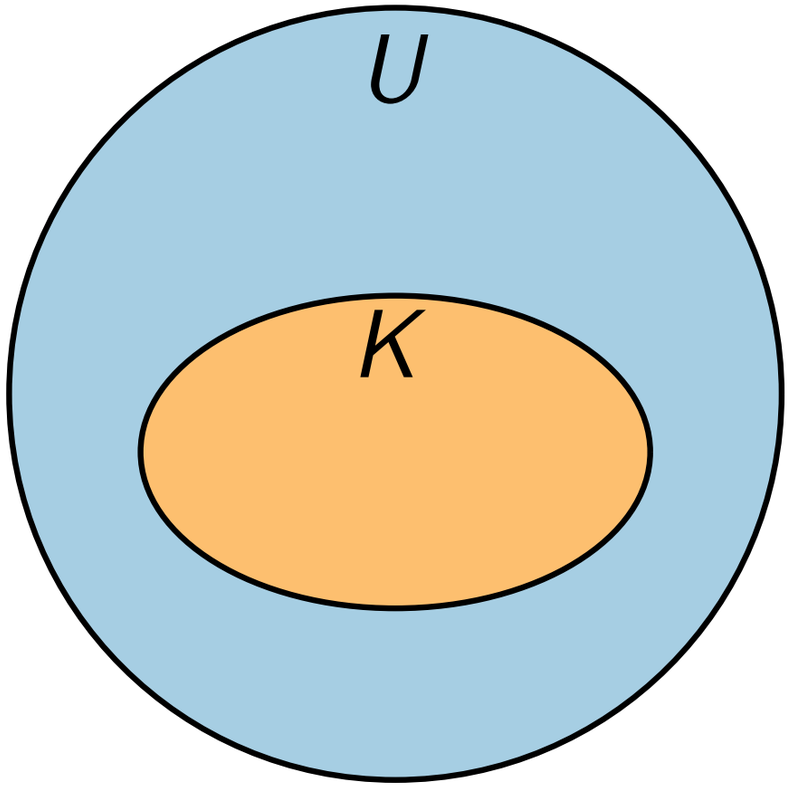
Strategie zur Kollisionsauflösung:

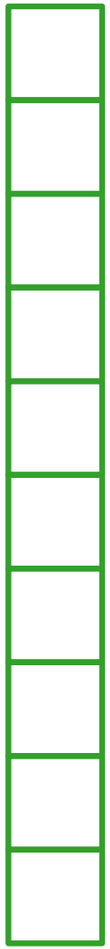
Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:



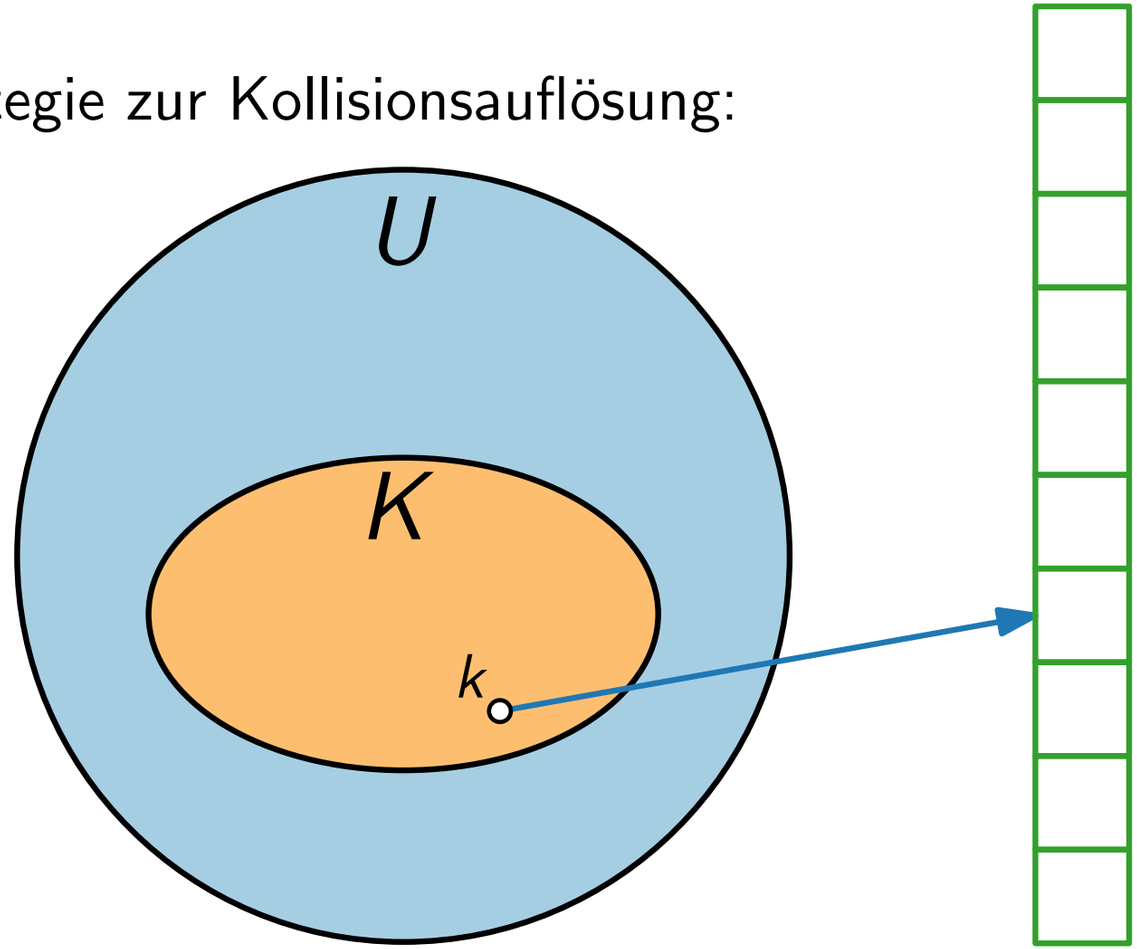


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

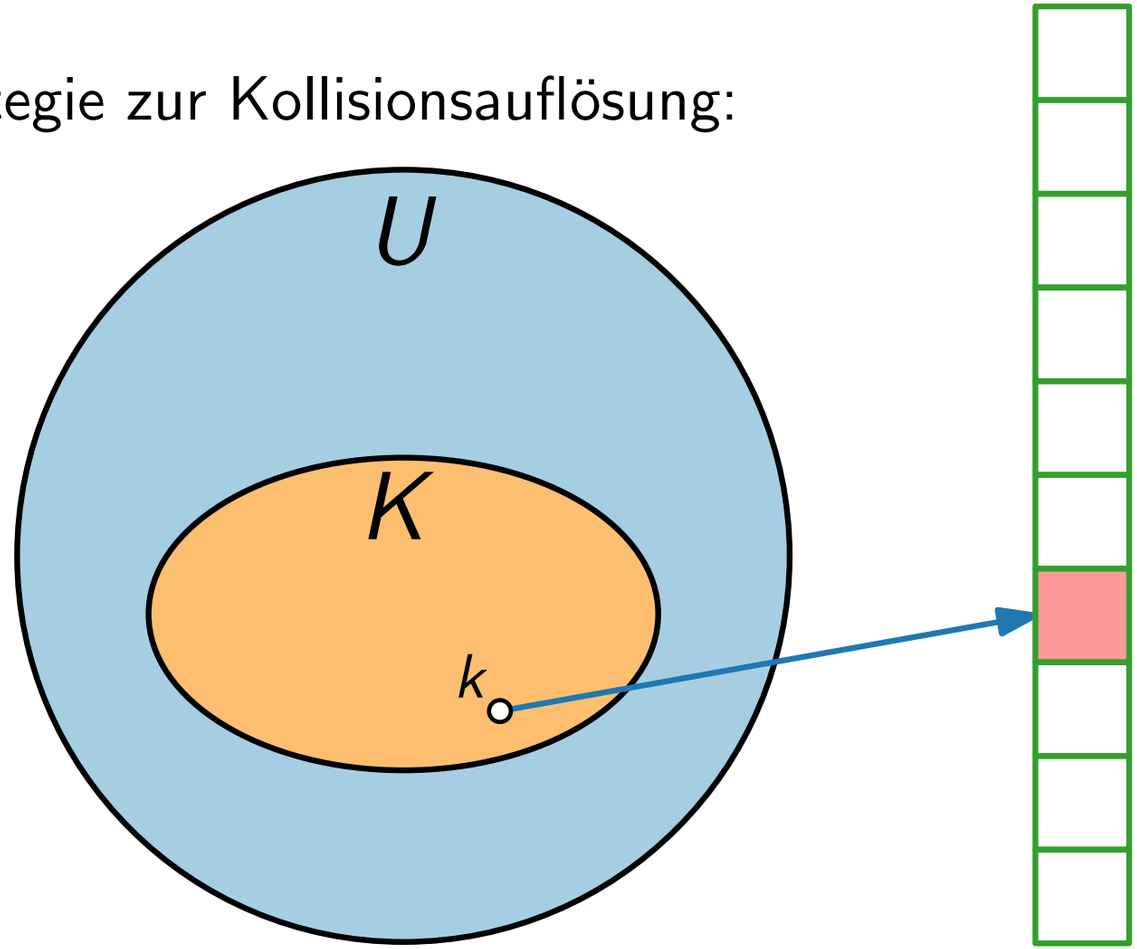


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

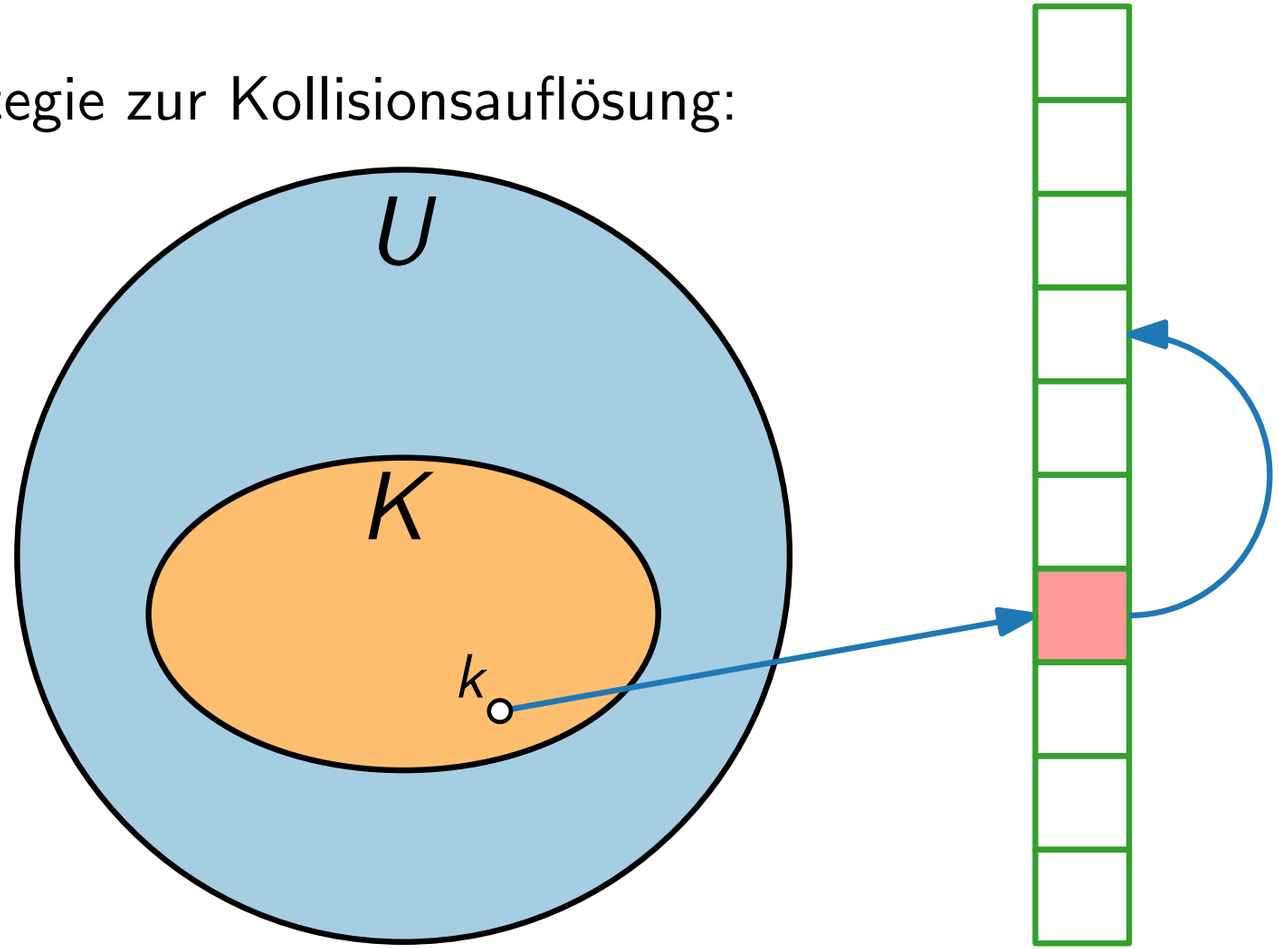


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

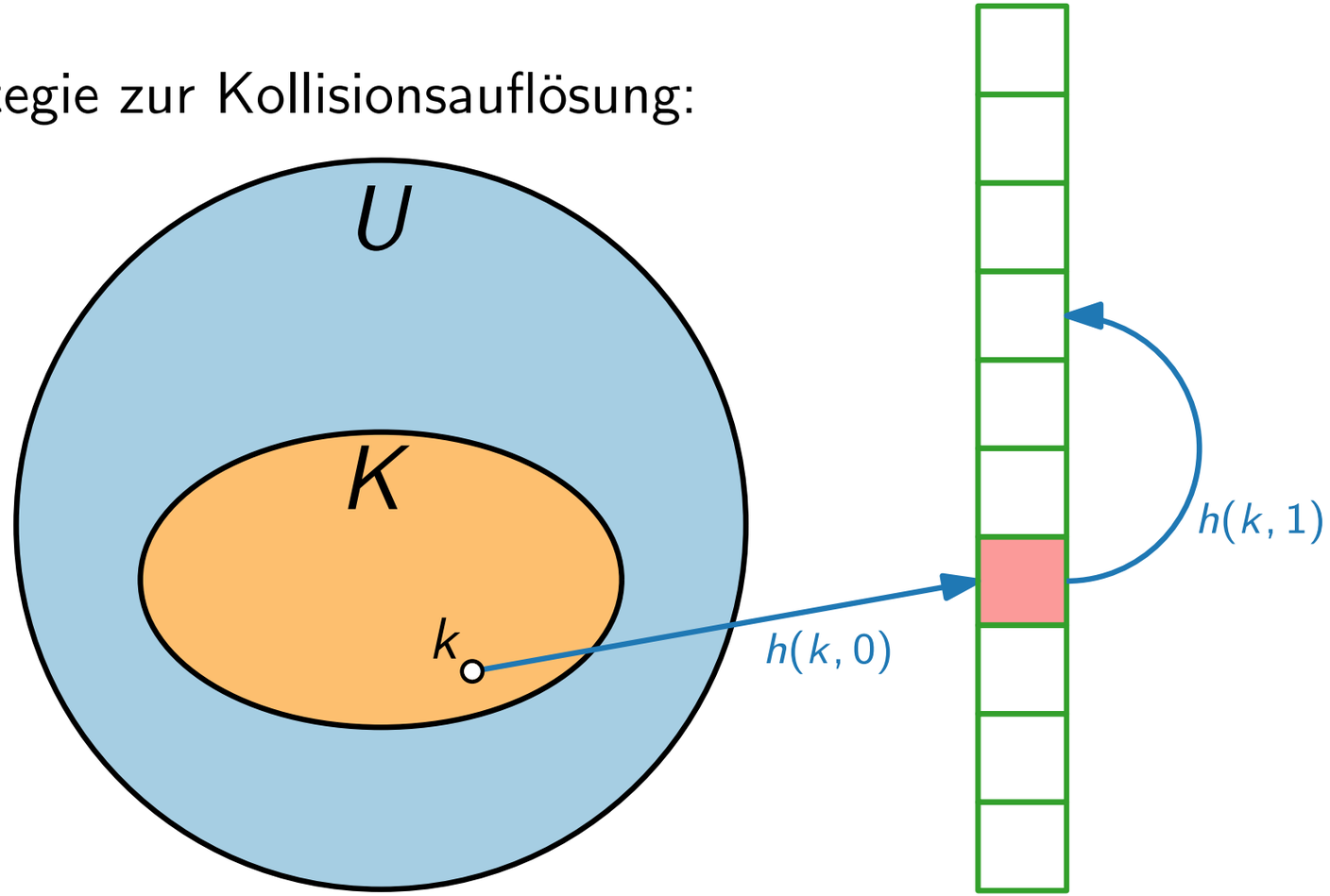


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

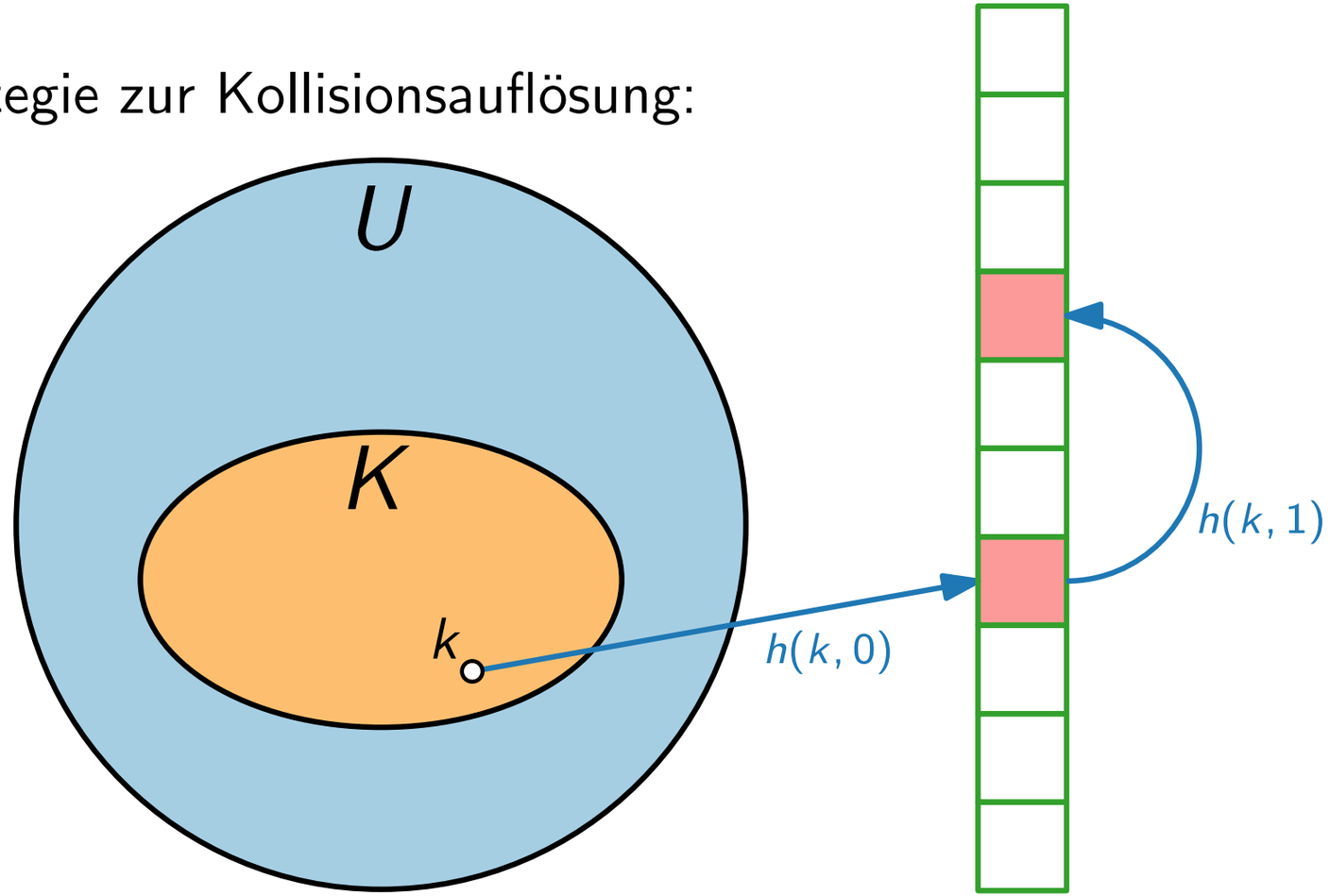


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

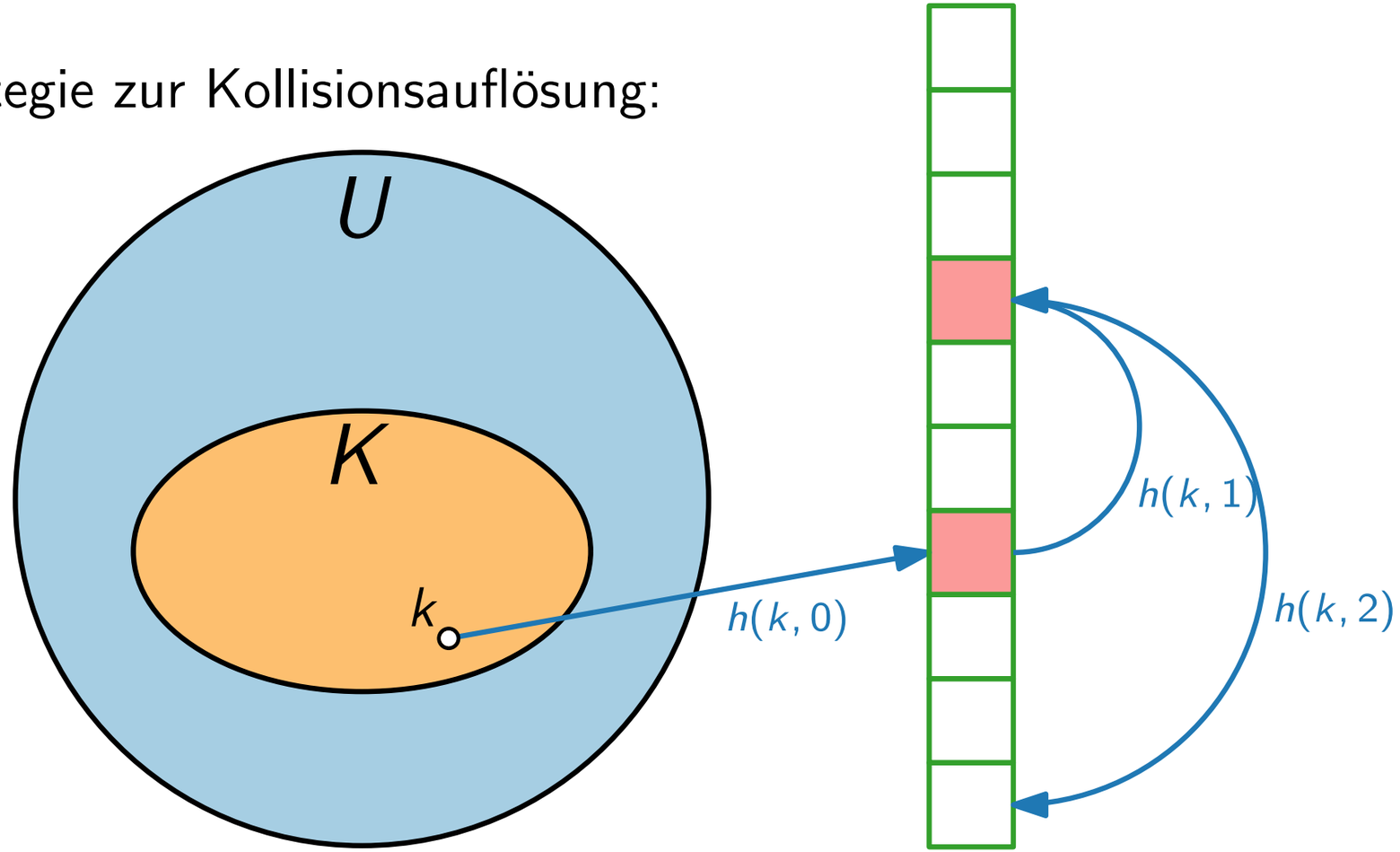


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

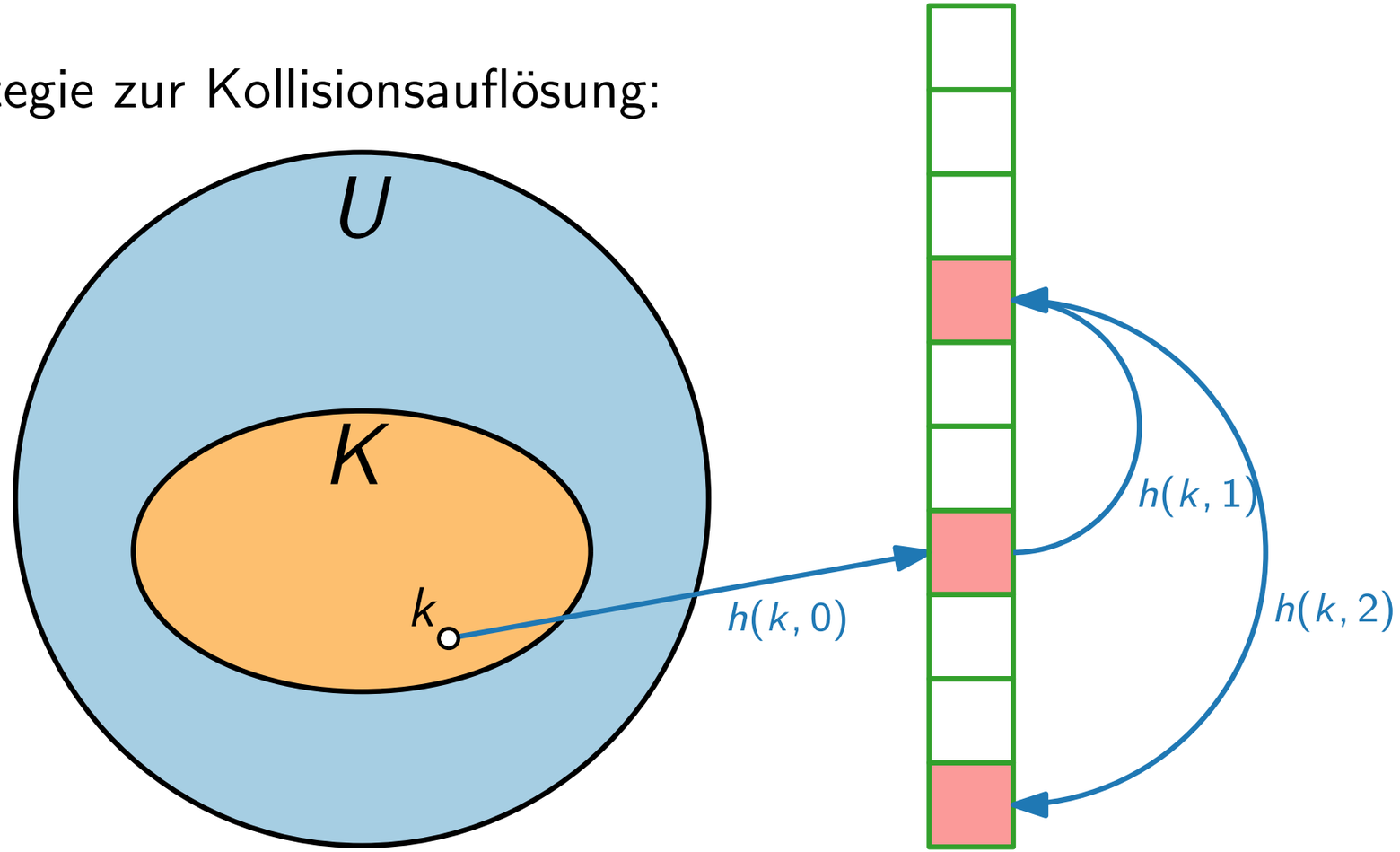


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

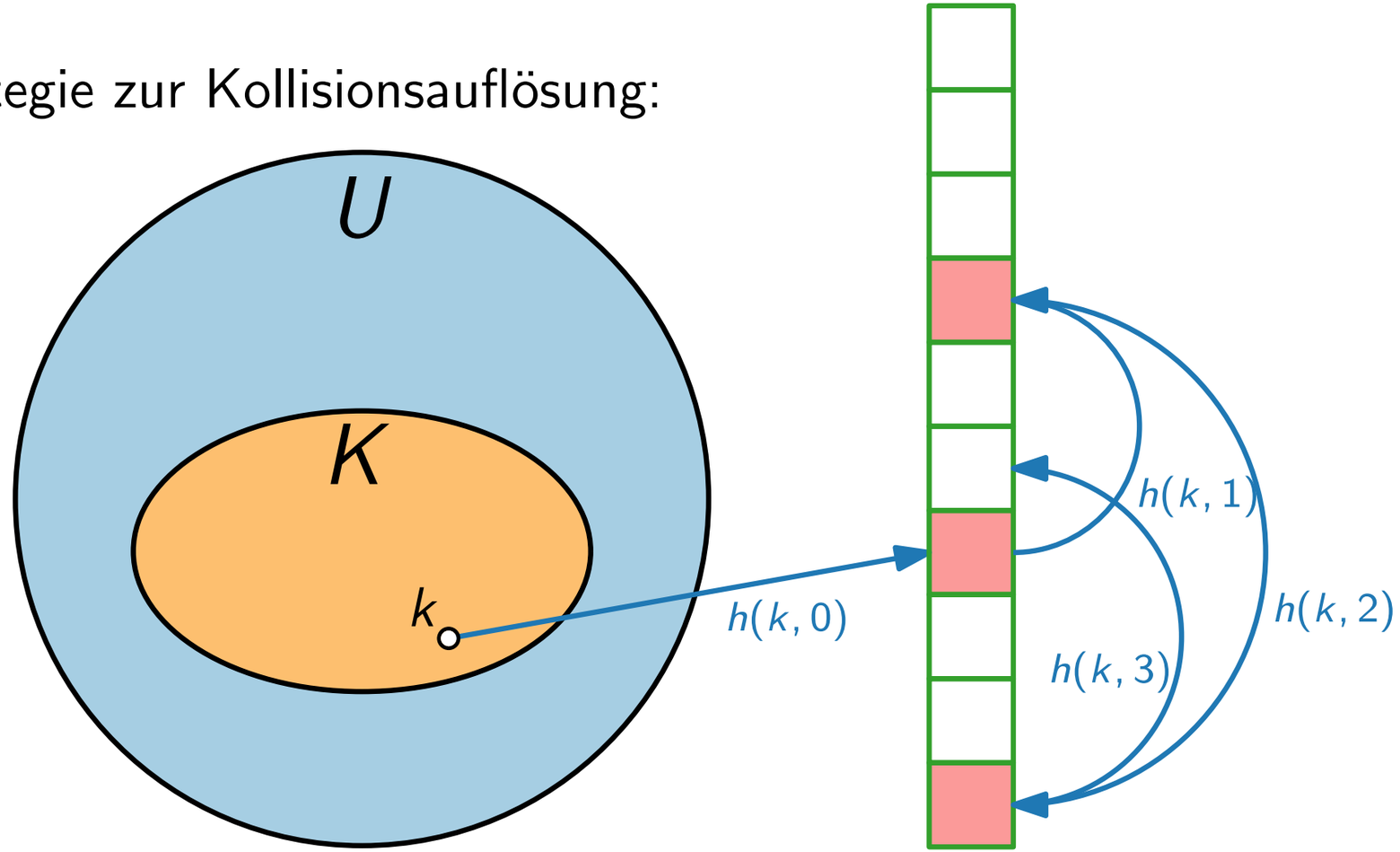


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

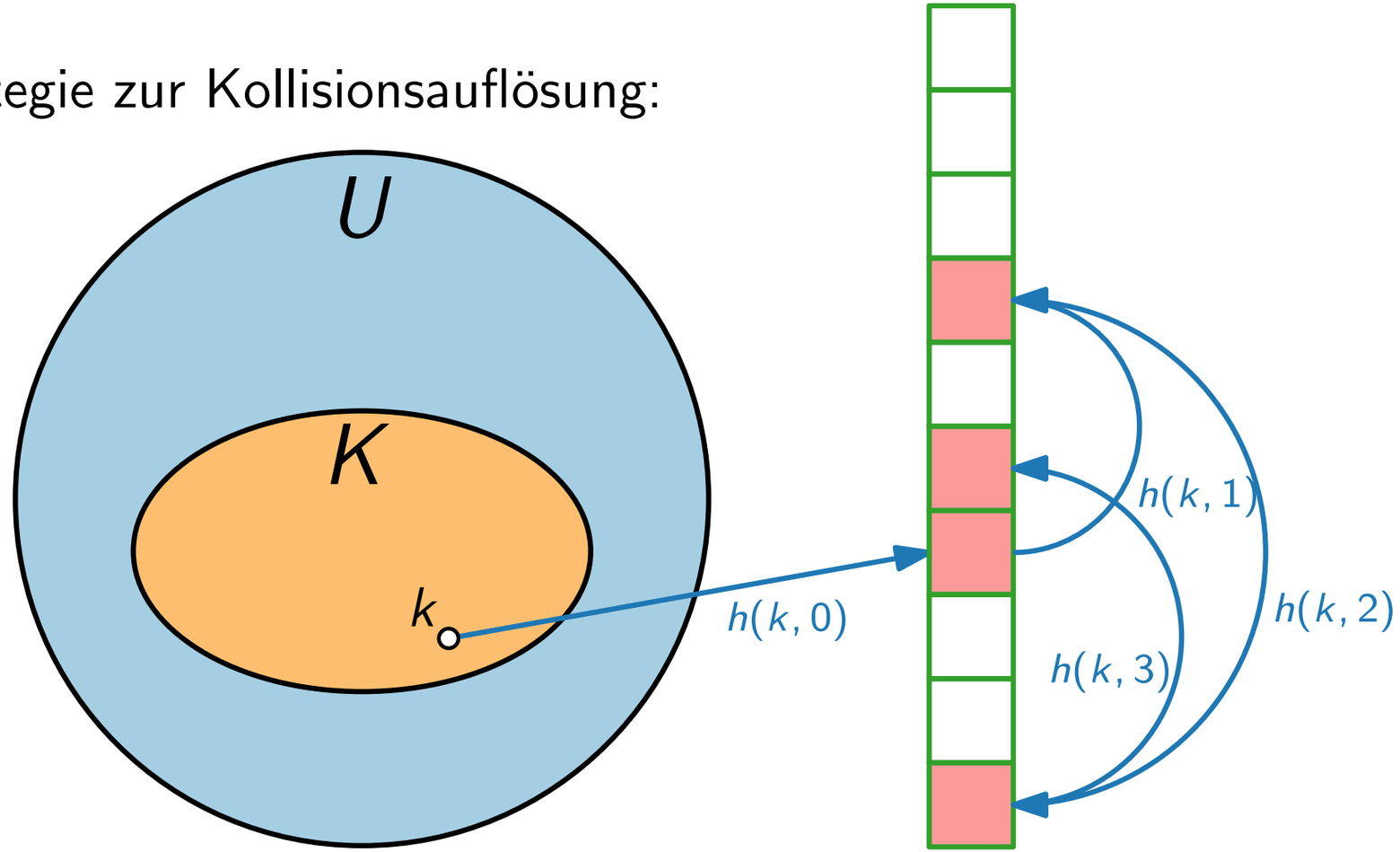


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

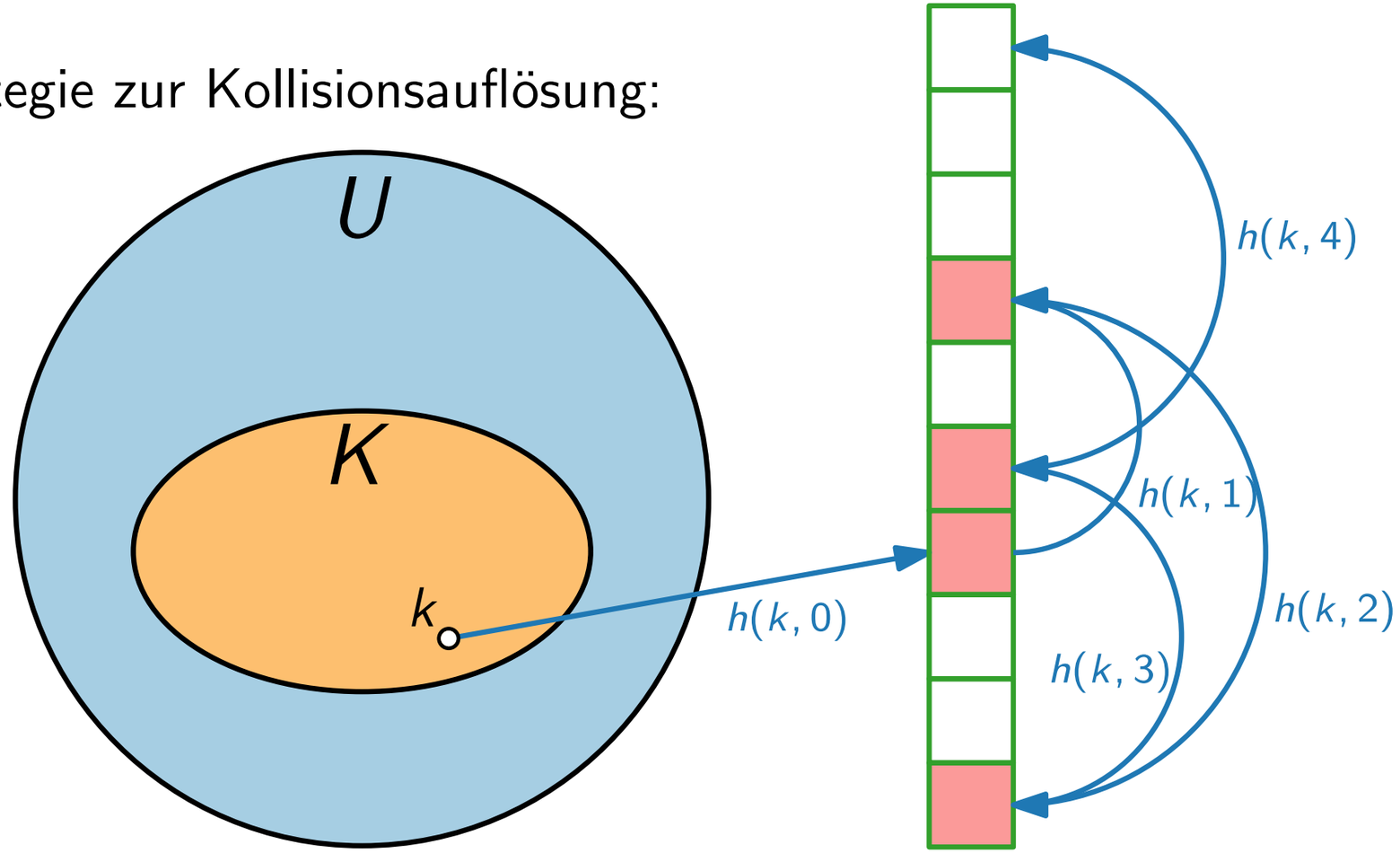


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

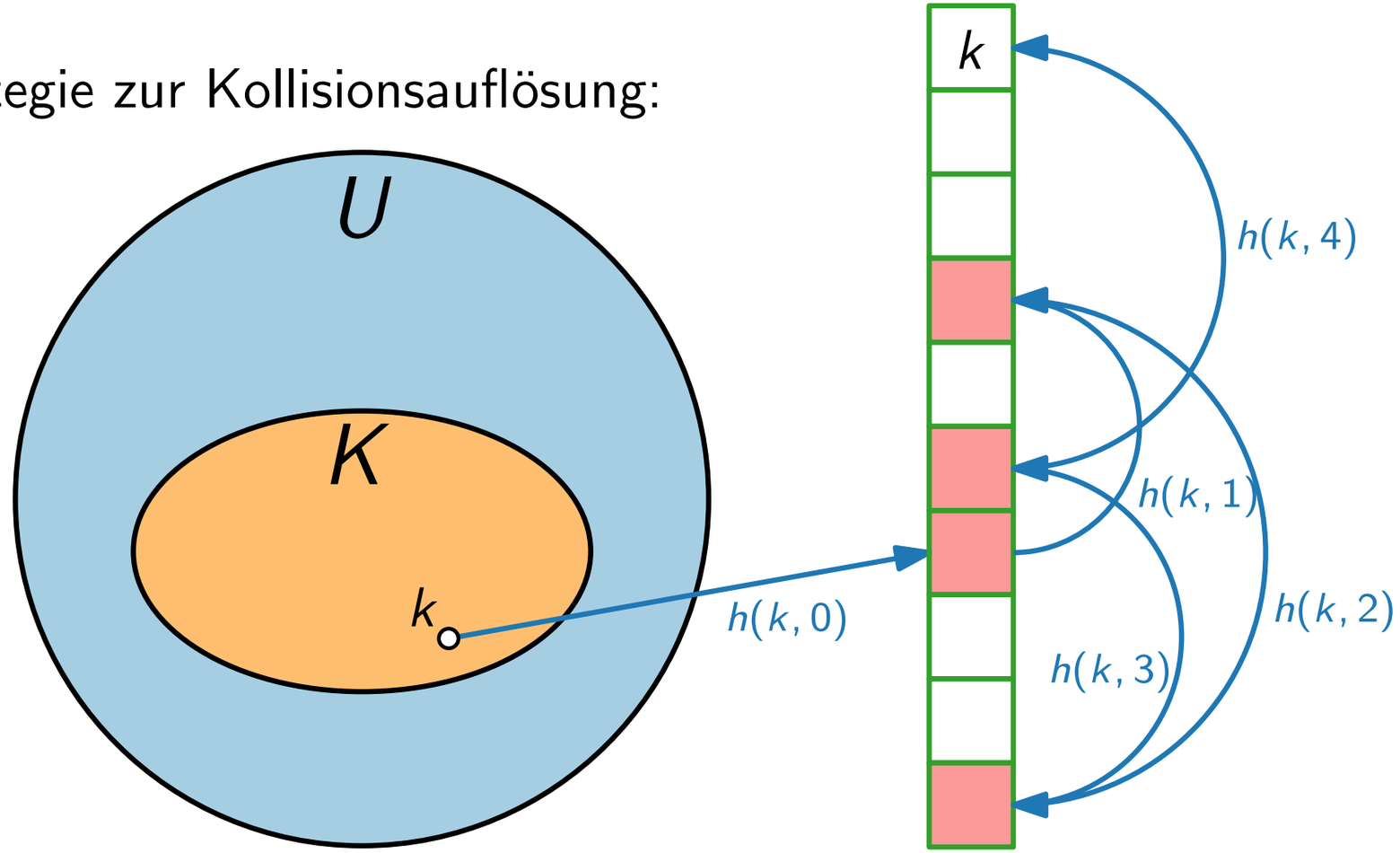


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

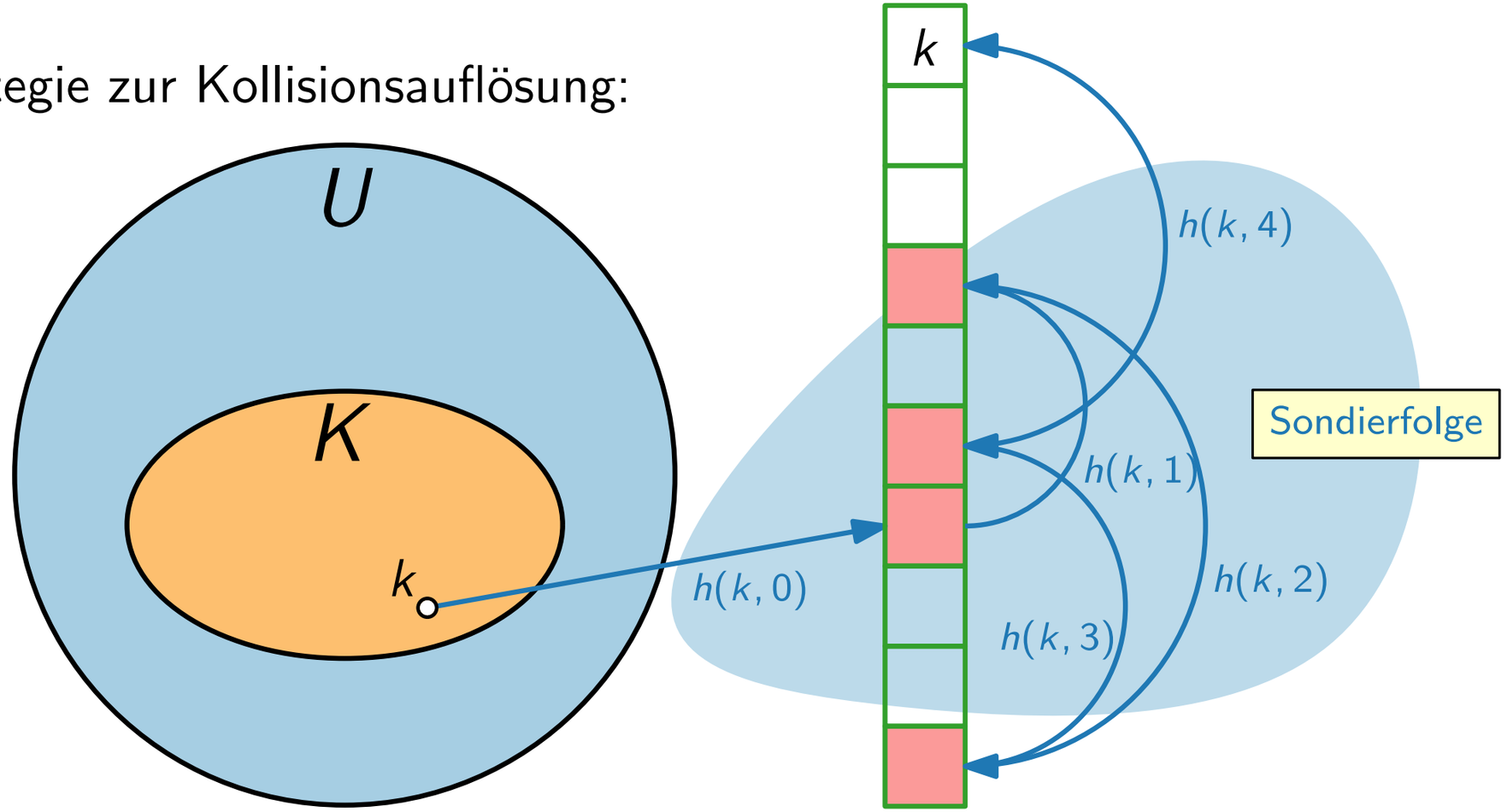


Hashing mit offener Adressierung

Alle Elemente werden direkt in der Hashtabelle gespeichert.

$\Rightarrow$ Tabelle kann volllaufen $\Rightarrow$ $\alpha \leq 1$

Strategie zur Kollisionsauflösung:

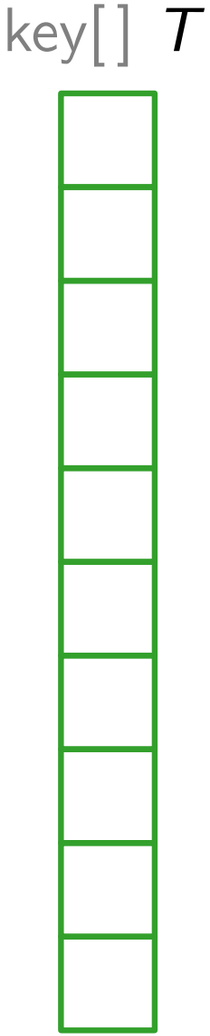


Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	
int Insert(key k)	
int Search(key k)	

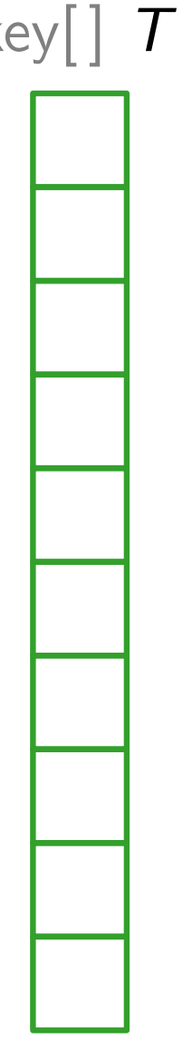
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	
int Insert(key k)	
int Search(key k)	



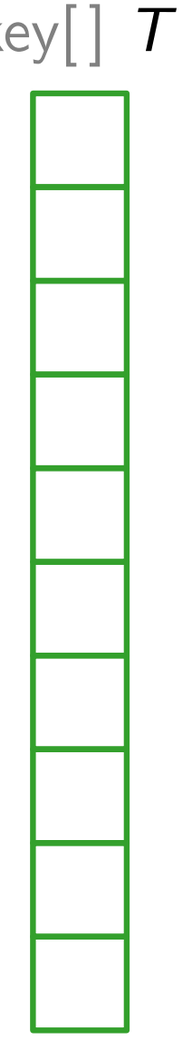
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	$T =$ **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$
int Insert(key k)	
int Search(key k)	



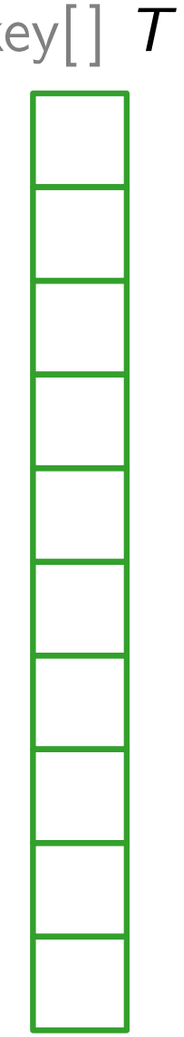
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	$T =$ **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$
int Insert(key k)	$i = 0$
int Search(key k)	



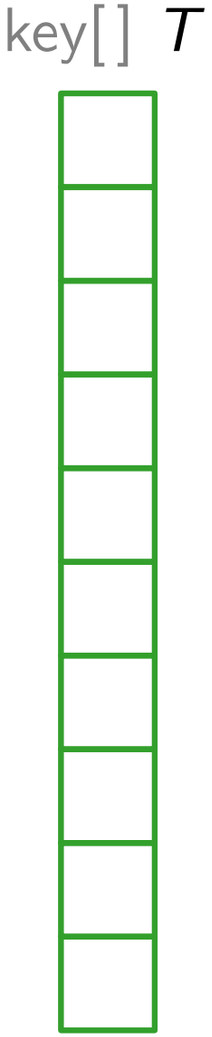
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$
int Insert(key k)	$i = 0$ **repeat**
	until $i == m$
int Search(key k)	



Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$
int Insert(key k)	$i = 0$ **repeat** Unterschied zu **while**? **until** $i == m$
int Search(key k)	

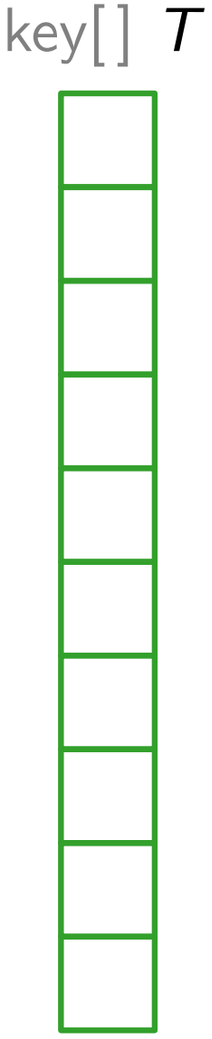


Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

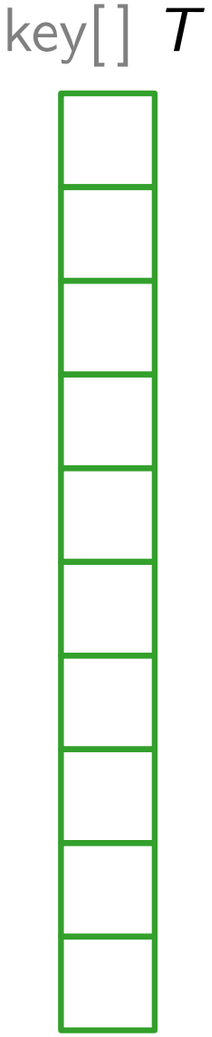
int Insert(key k)	$i = 0$
	repeat
	  Unterschied zu **while**?
	  Abbruchbedingung *nach* Ausführung des Schleifeninhalts
	  $\rightarrow$ Schleifeninhalt wird mindestens einmal ausgeführt
	until $i == m$

int Search(key k)	



Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$
int Insert(key k)	$i = 0$ **repeat** $j = h(k, i)$ **until** $i == m$
int Search(key k)	

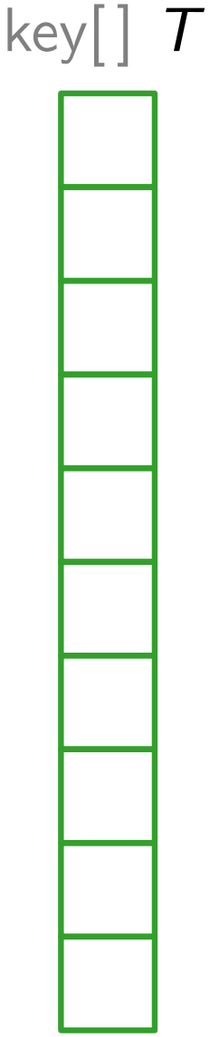


Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int Insert(key k)	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ **then**
	
	until $i == m$

int Search(key k)	

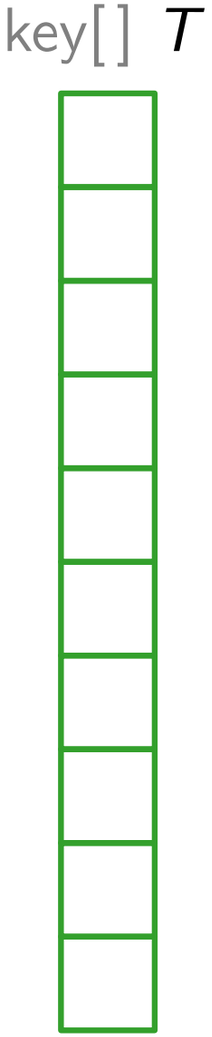


Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int Insert(key k)	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ **then**
	  $T[j] = k$
	  return j
	until $i == m$

int Search(key k)	

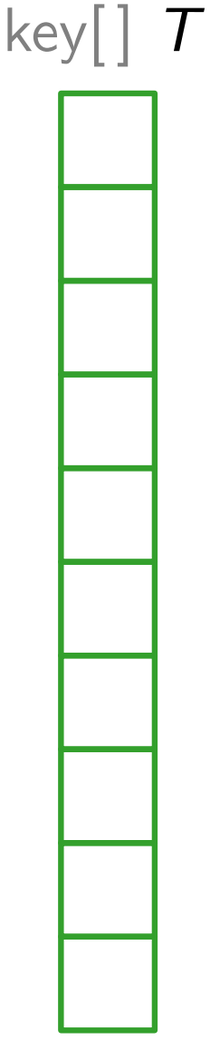


Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int Insert(key k)	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ **then**
	  $T[j] = k$
	  return j
	  else $i = i + 1$
	until $i == m$

int Search(key k)	

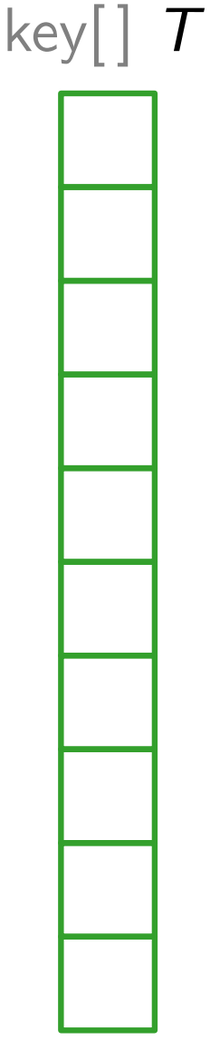


Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int Insert(key k)	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ **then**
	  $T[j] = k$
	  return j
	  else $i = i + 1$
	until $i == m$
	error "table overflow"

int Search(key k)	



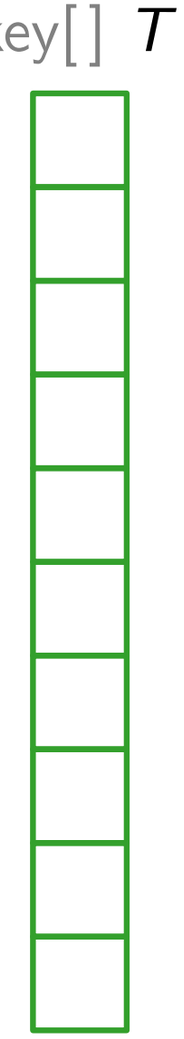
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int Insert(key k)	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ **then**
	  $T[j] = k$
	  return j
	  else $i = i + 1$
	until $i == m$
	error "table overflow"

int Search(key k)	

Aufgabe:
Schreiben Sie Search mit **repeat**-Schleife!



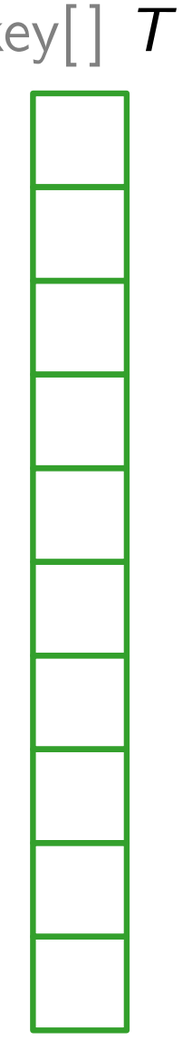
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int ~~Insert~~(key k)
Search
	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ **then**
	  $T[j] = k$
	  return j
	  else $i = i + 1$
	until $i == m$
	error "table overflow"

int Search(key k)	

Aufgabe:
Schreiben Sie Search mit **repeat**-Schleife!



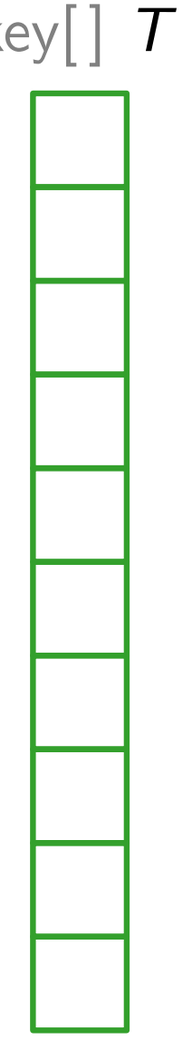
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int ~~Insert~~(key k)
Search
	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] ==$ ~~-1~~ **then**
	  ~~$T[j] = k$~~
	  return j
	  else $i = i + 1$
	until $i == m$
	error "table overflow"

Aufgabe:
Schreiben Sie Search mit **repeat**-Schleife!

int Search(key k)



Hashing mit offener Adressierung

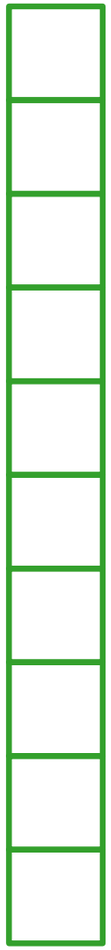
Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int ~~Insert~~(key k)
Search
	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ ~~k~~ **then**
	  ~~$T[j] = k$~~
	  return j
	  else $i = i + 1$
	until $i == m$
	error "table overflow"

int Search(key k)	

Aufgabe:
Schreiben Sie Search mit **repeat**-Schleife!

key[ ] T



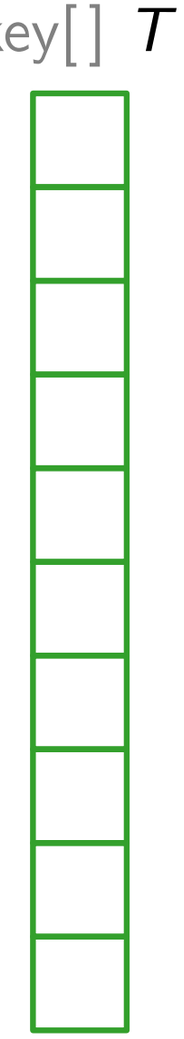
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \ldots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int ~~Insert~~(key k)
Search
	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ **then** (k)
	  $T[j] = k$
	  return j
	  else $i = i + 1$
	until $i == m$ **or**
	error "table overflow"

int Search(key k)	

Aufgabe:
Schreiben Sie Search mit **repeat**-Schleife!



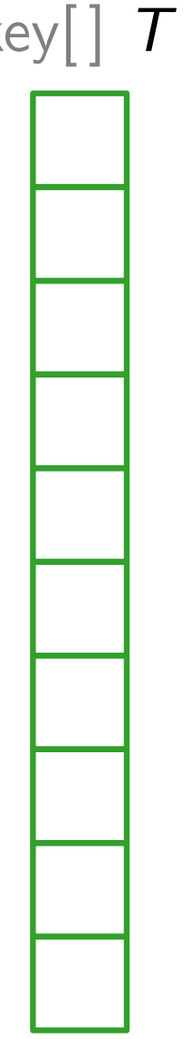
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int ~~Insert~~(key k)
Search
	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == ~~-1~~$ k **then**
	  ~~$T[j] = k$~~
	  return j
	  else $i = i + 1$
	until $i == m$ **or** $T[j] == -1$
	error "table overflow"

int Search(key k)	

Aufgabe:
Schreiben Sie Search mit **repeat**-Schleife!



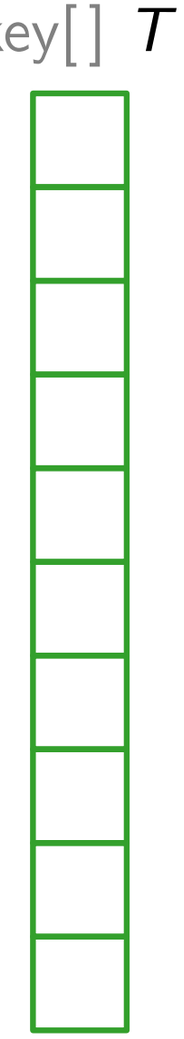
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int ~~Insert~~(key k)
Search
	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] == -1$ ~~k~~ **then**
	  ~~$T[j] = k$~~
	  return j
	  else $i = i + 1$
	until $i == m$ **or** $T[j] == -1$
	~~**error** "table overflow"~~ **return** -1

int Search(key k)	

Aufgabe:
Schreiben Sie Search mit **repeat**-Schleife!



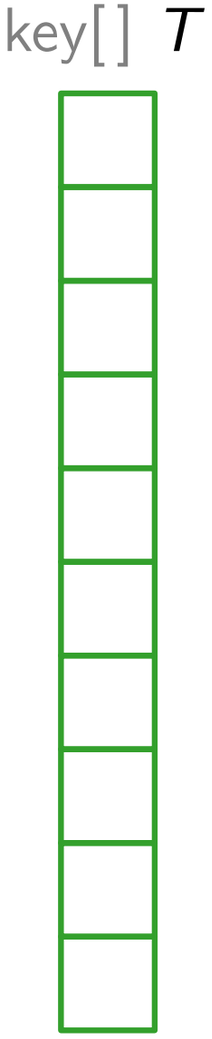
Hashing mit offener Adressierung

Operation	Implementierung
HashOA(int m)	T = **new** key$[0 \dots m-1]$ **for** $j = 0$ **to** $m-1$ **do** $T[j] = -1$

int ~~Insert~~(key k)
Search
	$i = 0$
	repeat
	  $j = h(k, i)$
	  if $T[j] ==$ ~~-1~~ k **then**
	  ~~$T[j] = k$~~
	  return j
	  else $i = i + 1$
	until $i == m$ **or** $T[j] == -1$
	~~**error** "table overflow"~~ **return** -1

int Search(key k)	

...und Delete()?



Hashing mit offener Adressierung

Operation

HashOA(int m)

int ~~Insert~~(key k)

Search

...und Delete()?

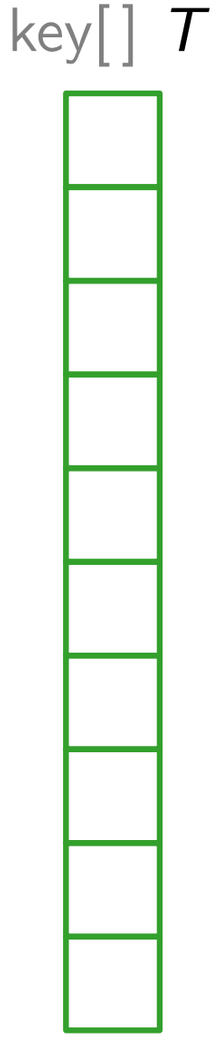
Umständlich; z.B. indem man **Grabsteine** hinterlässt, die einem sagen, dass man bei Search weitersuchen soll.

int Search(key k)

Implementierung

T = **new** key$[0 \ldots m-1]$
for $j = 0$ **to** $m-1$ **do** $T[j] = -1$

$i = 0$
repeat
  $j = h(k, i)$
  if $T[j] == $ ~~-1~~ k **then**
    ~~$T[j] = k$~~
    return j
  else $i = i + 1$
until $i == m$ **or** $T[j] == -1$
~~**error** "table overflow"~~ **return** -1



Hashing mit offener Adressierung

Operation

HashOA(int m)

int ~~Insert~~(key k)

Search

...und Delete()?

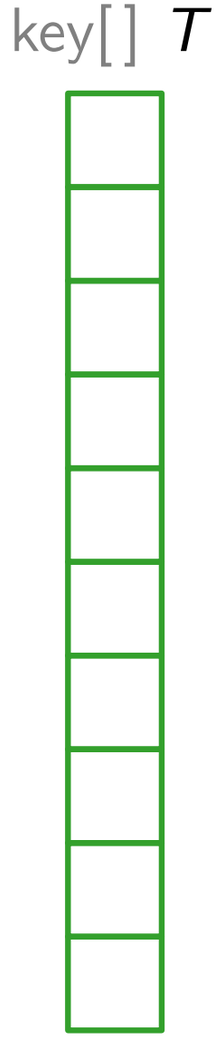
Umständlich; z.B. indem man **Grabsteine** hinterlässt, die einem sagen, dass man bei Search weitersuchen soll.

int Search(key k)

Implementierung

T = **new** key$[0 \dots m-1]$
for $j = 0$ **to** $m-1$ **do** $T[j] = -1$

$i = 0$
repeat
  $j = h(k, i)$
  if $T[j] == ~~-1~~$ k **then**
    ~~$T[j] = k$~~
    return j
  else $i = i + 1$
until $i == m$ **or** $T[j] == -1$
~~**error** "table overflow"~~ **return** -1



Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h :

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$

  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$

  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$

  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Verschiedene Typen von Sondierfolgen:

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$

  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Verschiedene Typen von Sondierfolgen:

- Lineares Sondieren:

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$

  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Verschiedene Typen von Sondierfolgen:

- Lineares Sondieren: $\quad h(k,i) = (h_0(k) + i) \bmod m$

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$
  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Verschiedene Typen von Sondierfolgen:

- Lineares Sondieren: $h(k,i) = (h_0(k) + i) \bmod m$
- Quadratisches Sondieren:

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$
  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Verschiedene Typen von Sondierfolgen:

- Lineares Sondieren: $\quad h(k,i) = (h_0(k) + i) \bmod m$
- Quadratisches Sondieren: $\quad h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$
  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Verschiedene Typen von Sondierfolgen:

- Lineares Sondieren: $\quad h(k,i) = (h_0(k) + i) \bmod m$
- Quadratisches Sondieren: $\quad h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$
- Doppeltes Hashing:

Berechnung von Sondierfolgen

Hashfunktion für offene Adressierung h : $\quad U \times \{0, \ldots, m-1\} \to \{0, \ldots, m-1\}$

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ heißt **Sondierfolge** (für k).

Voraussetzungen:

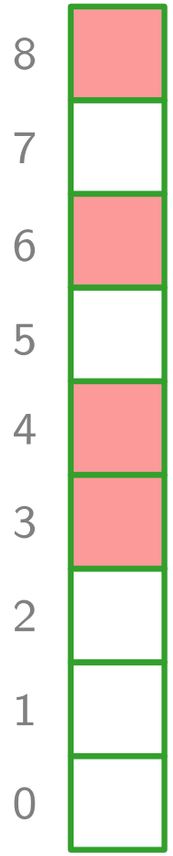
- eine Sondierfolge ist eine Permutation von $\langle 0, 1, \ldots, m-1 \rangle$
  (Sonst durchläuft die Folge nicht alle Tabelleneinträge genau einmal!)
- Existenz von „gewöhnlicher“ Hashfunktion $h_0 \colon U \to \{0, \ldots, m-1\}$

Verschiedene Typen von Sondierfolgen:

- Lineares Sondieren: $\quad h(k,i) = (h_0(k) + i) \bmod m$
- Quadratisches Sondieren: $\quad h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$
- Doppeltes Hashing: $\quad h(k,i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Lineares Sondieren

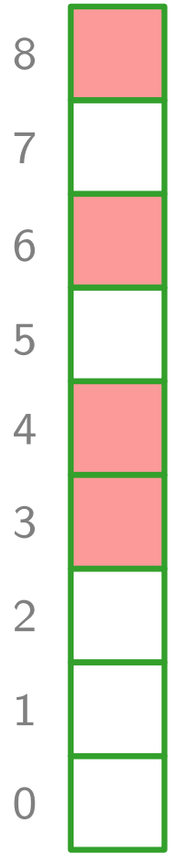
Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$



Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

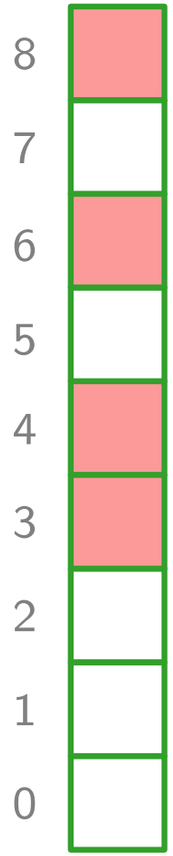


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

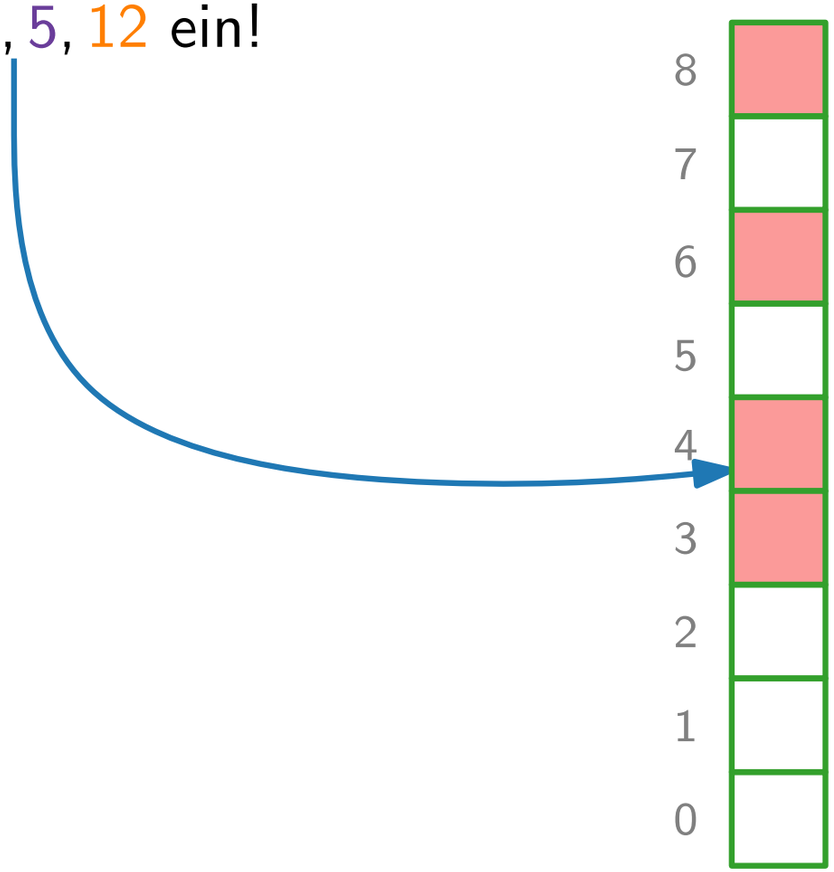


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

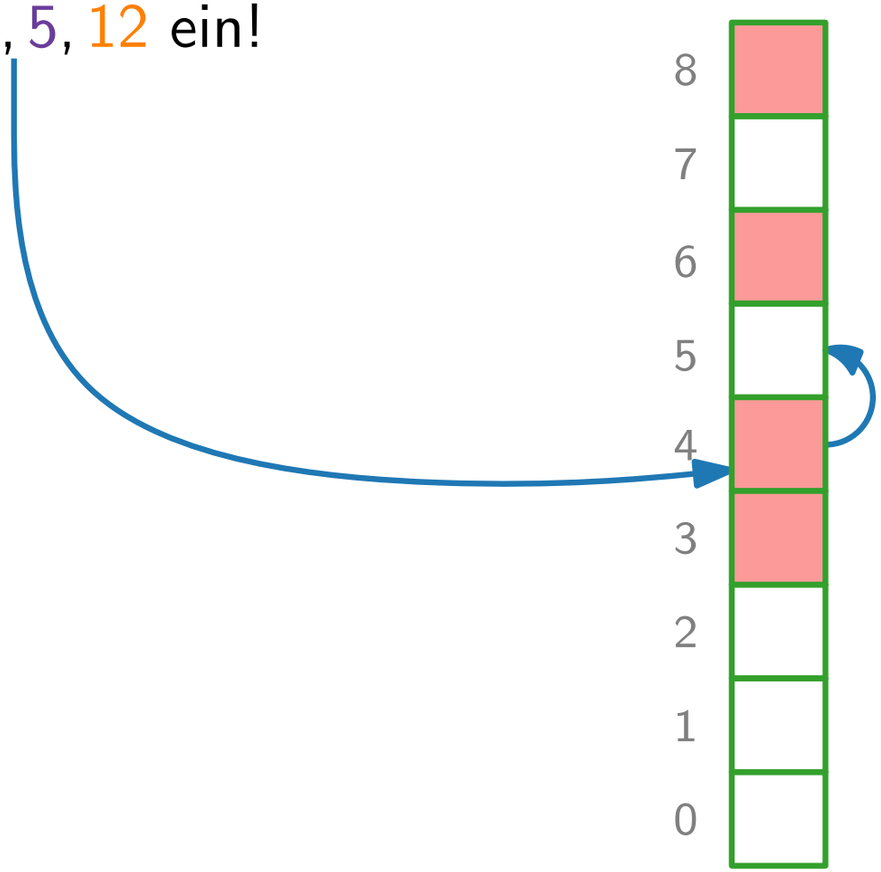


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

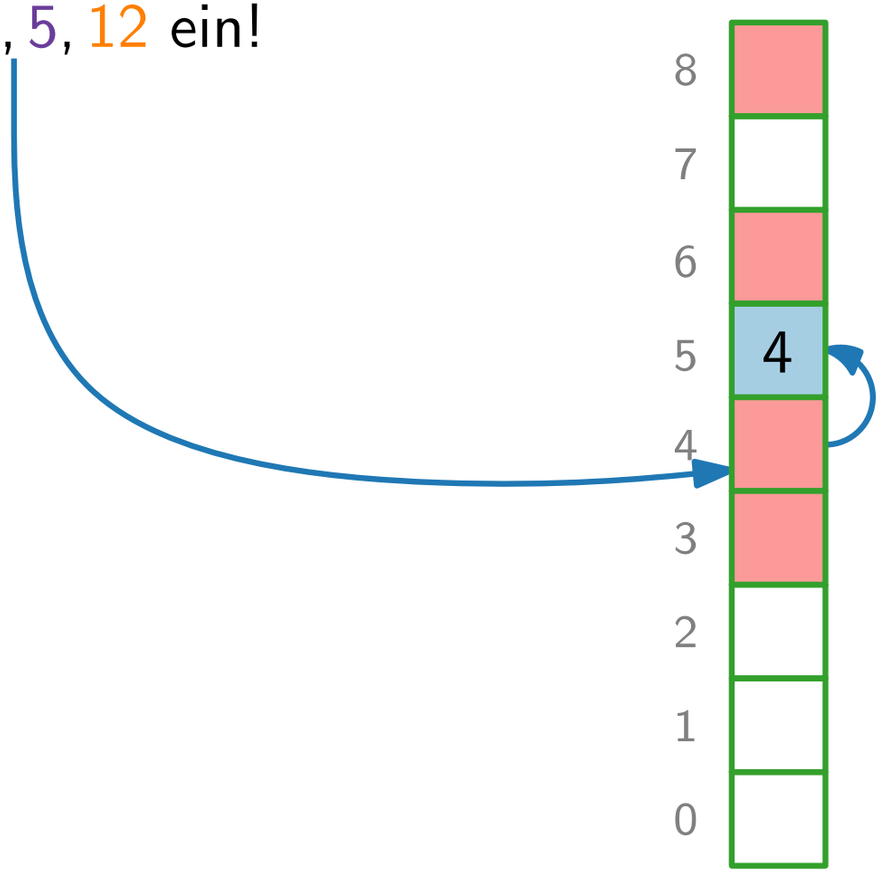


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

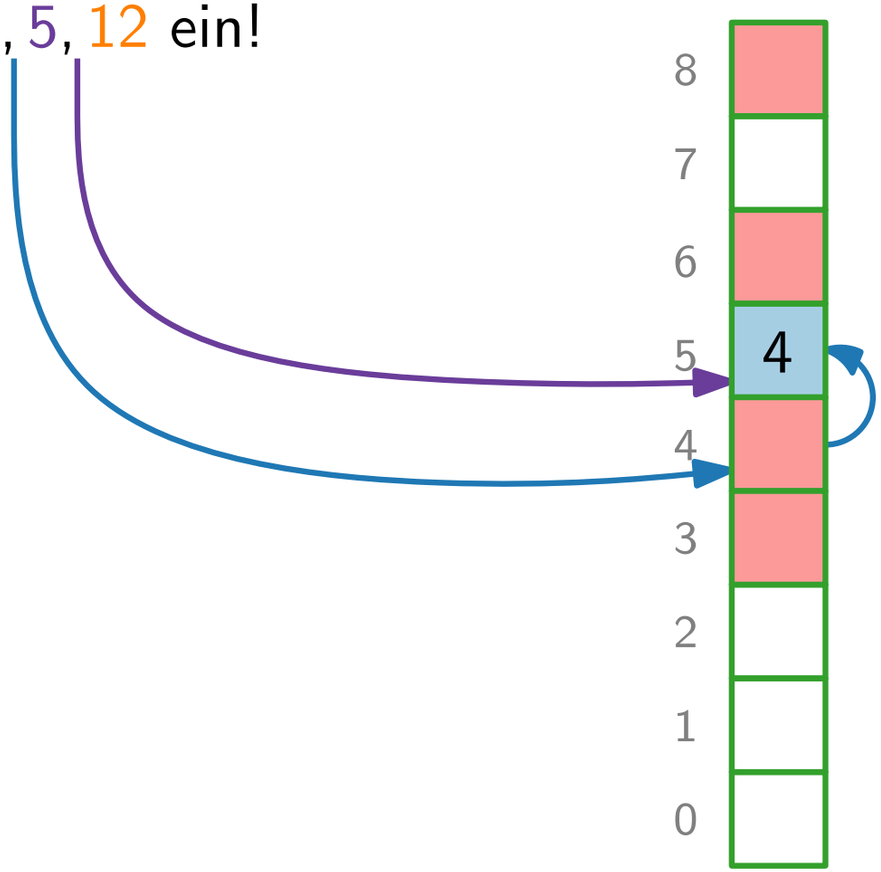


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

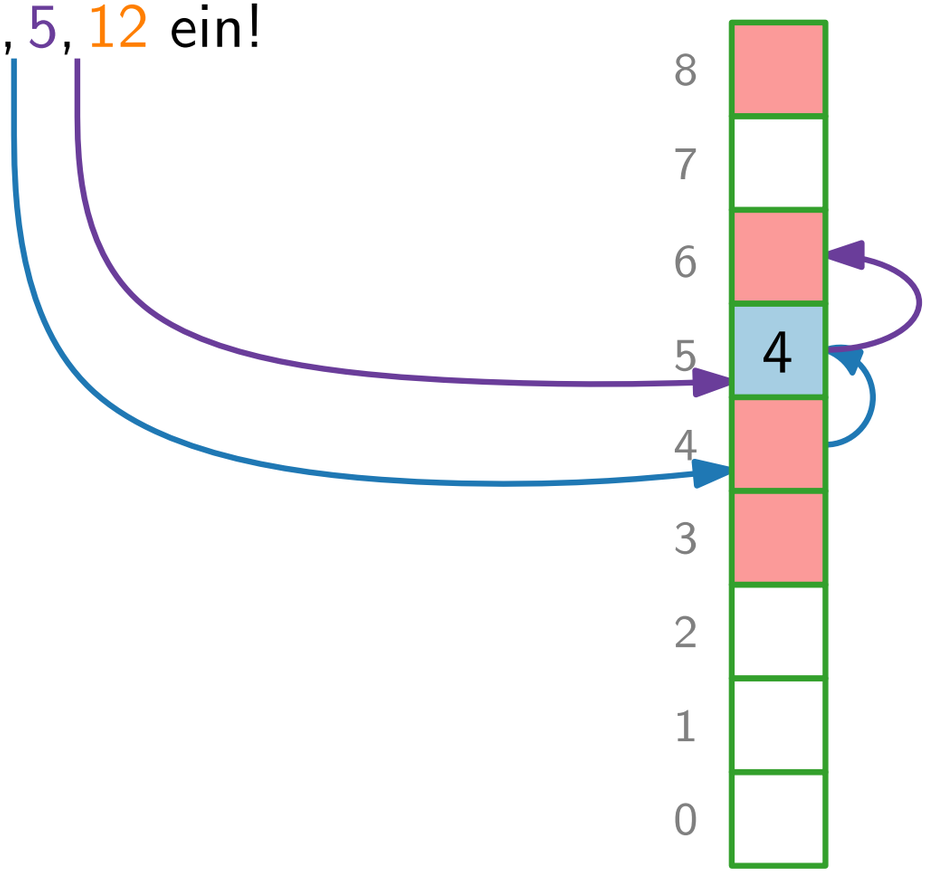


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

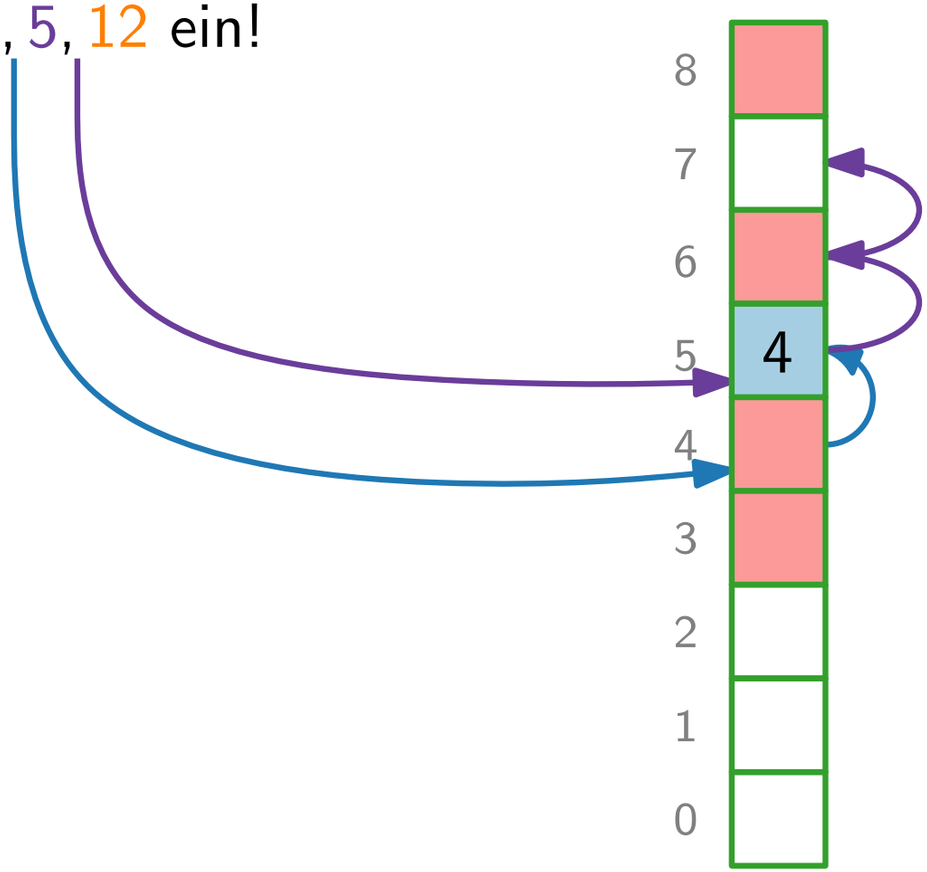


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

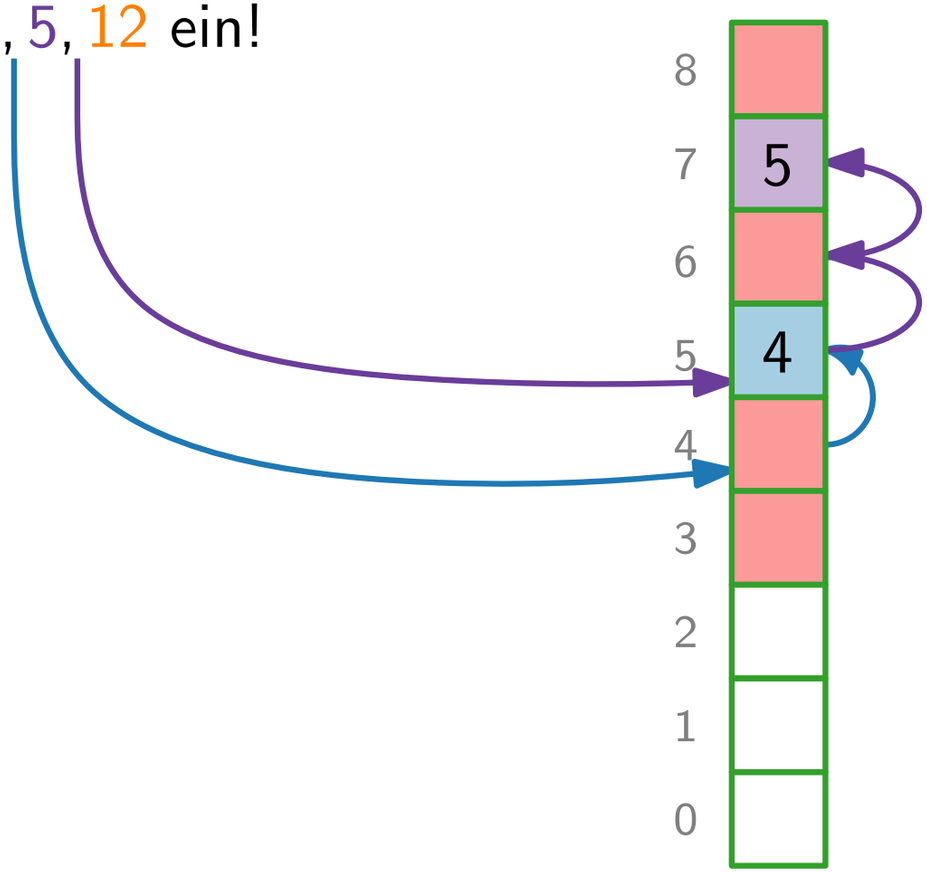


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

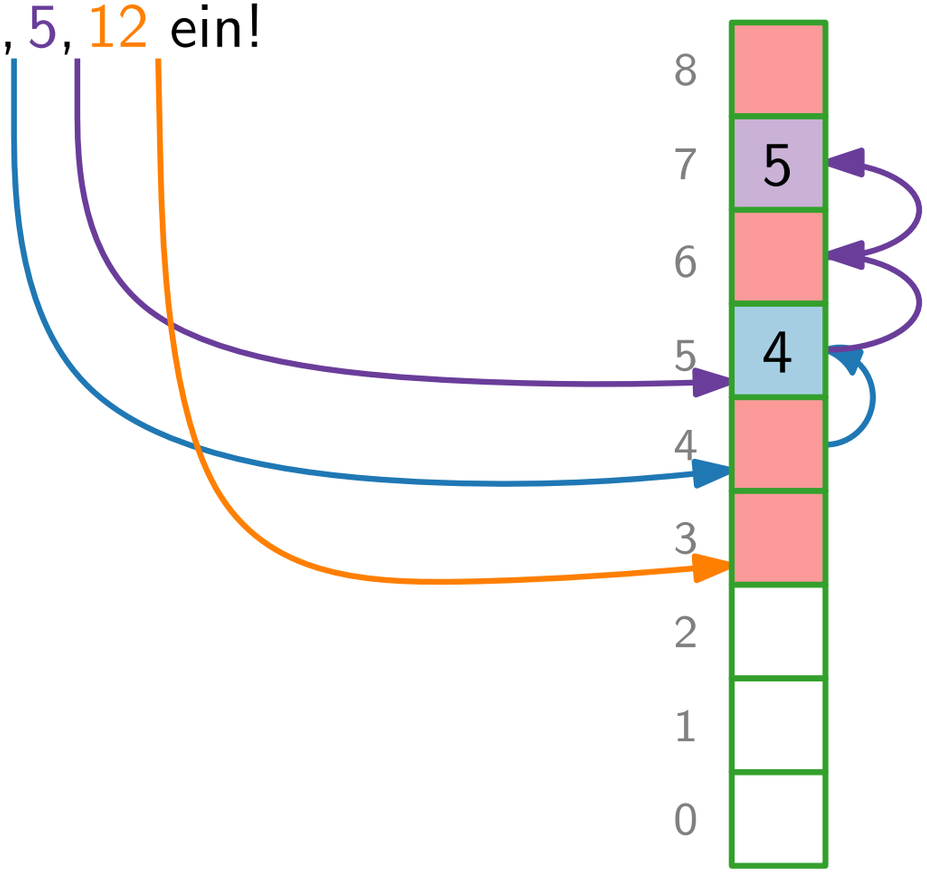


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

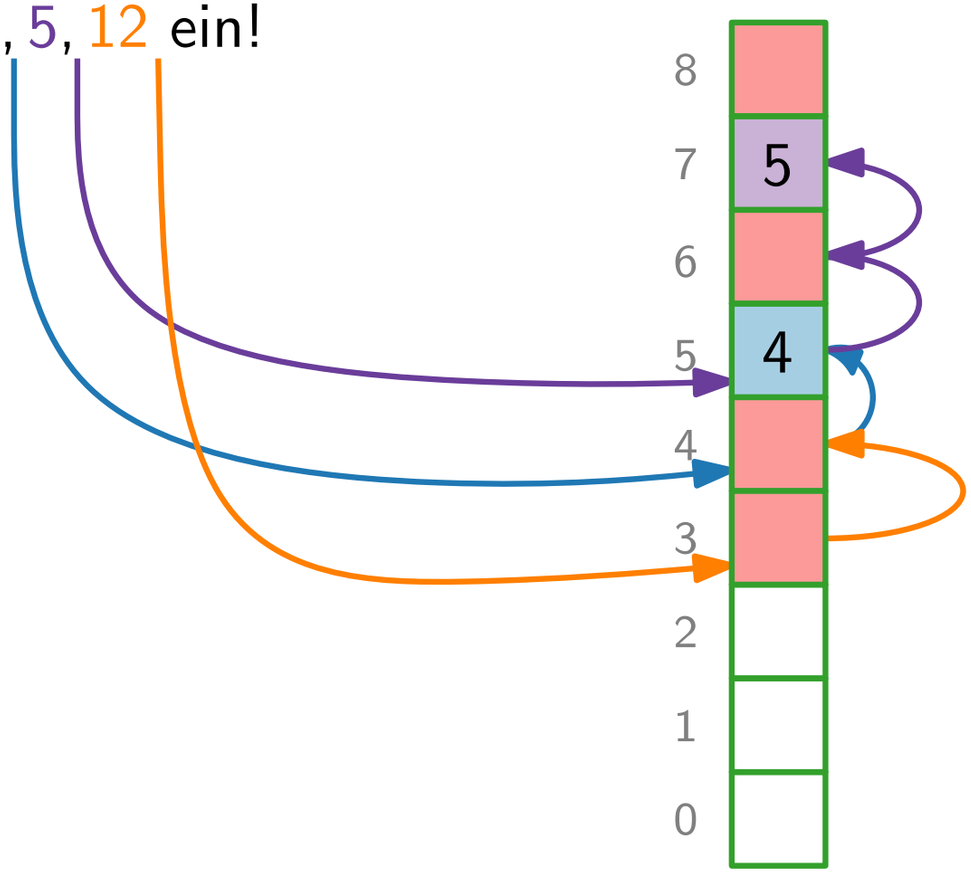


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

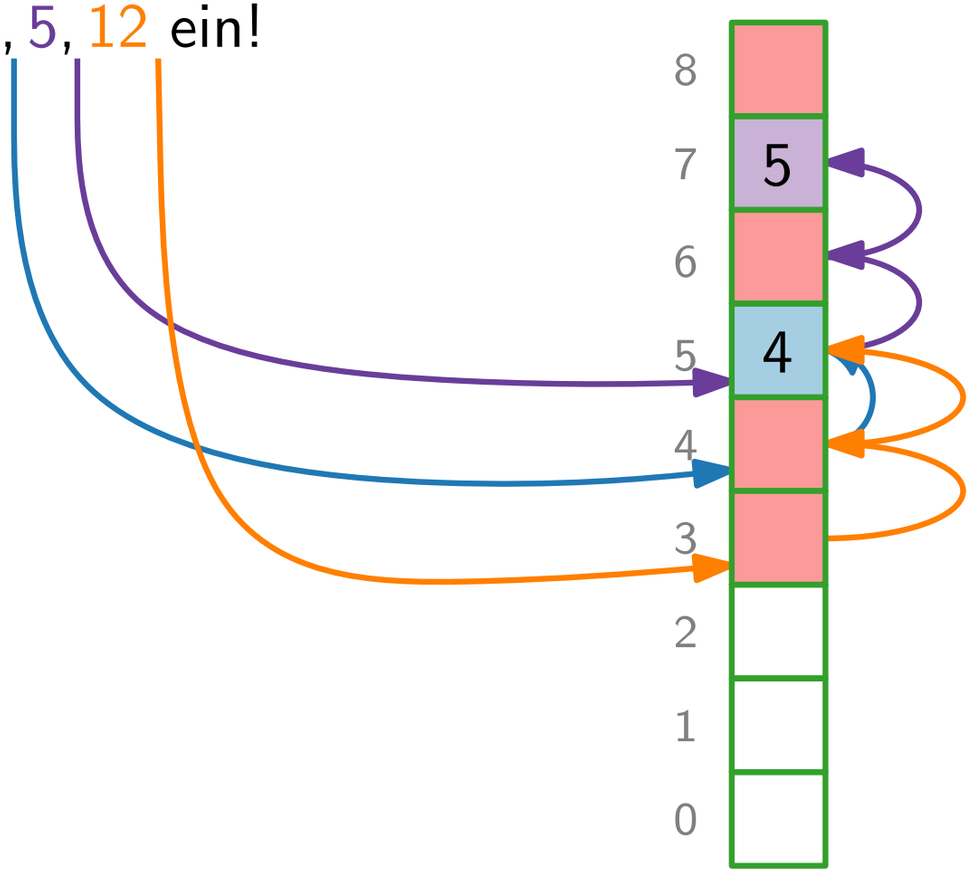


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

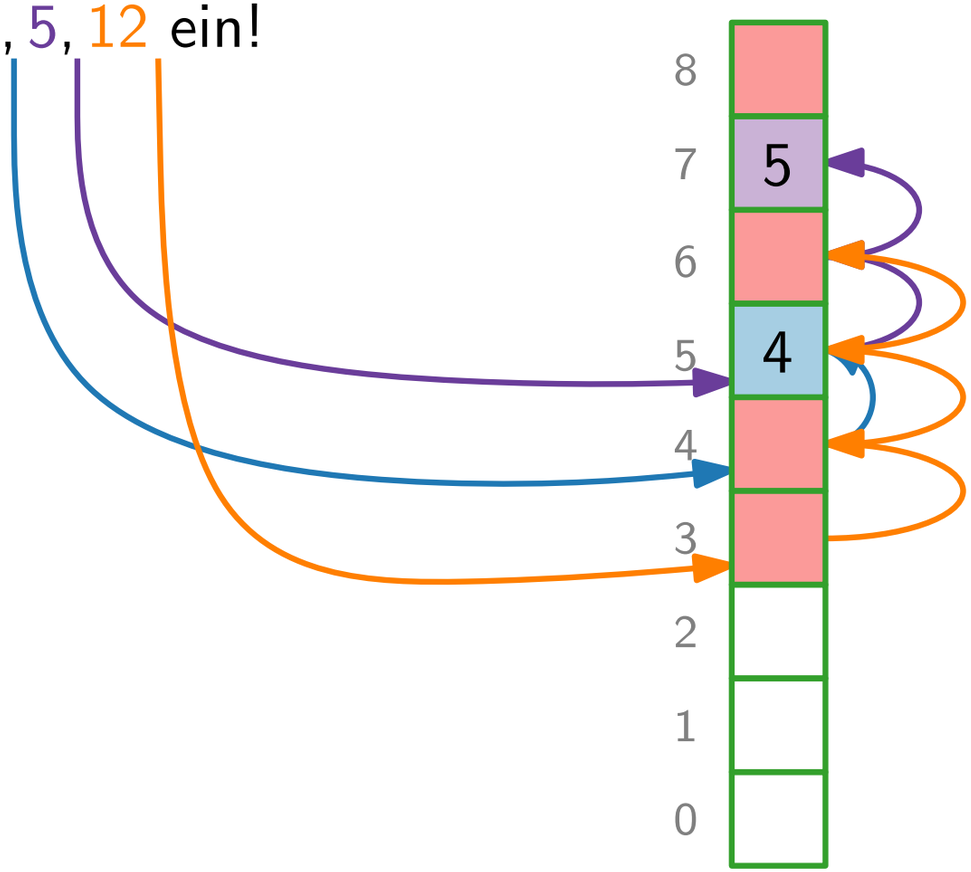


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

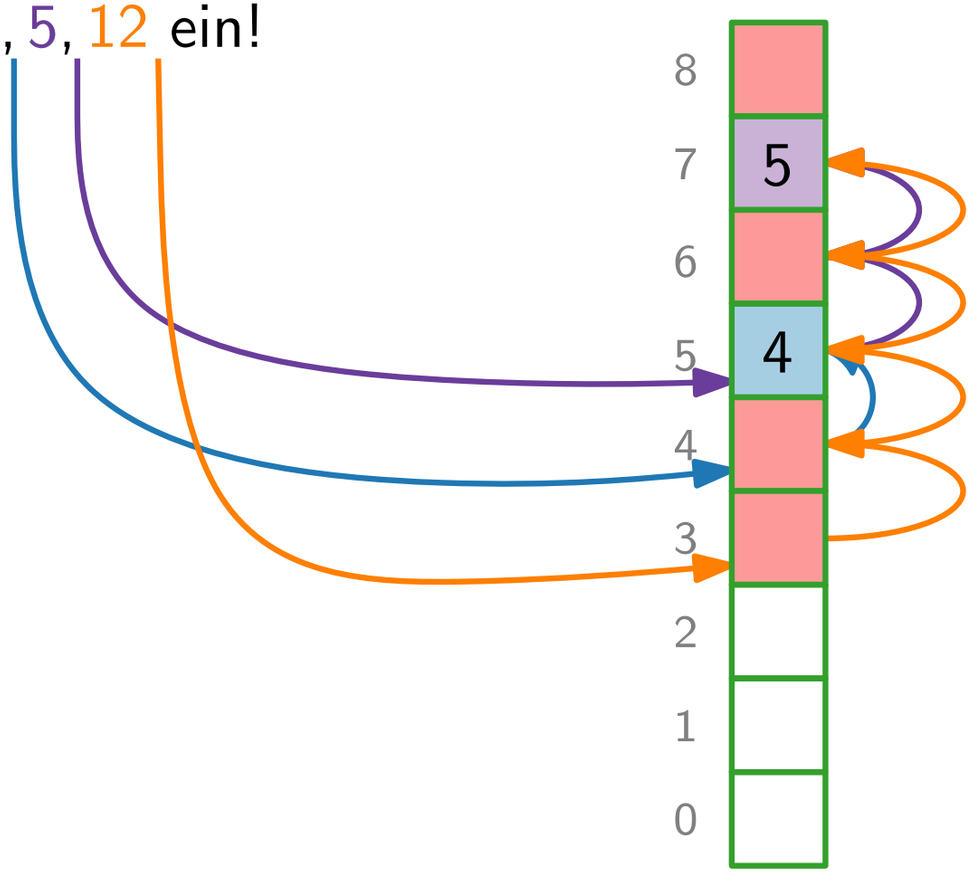


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

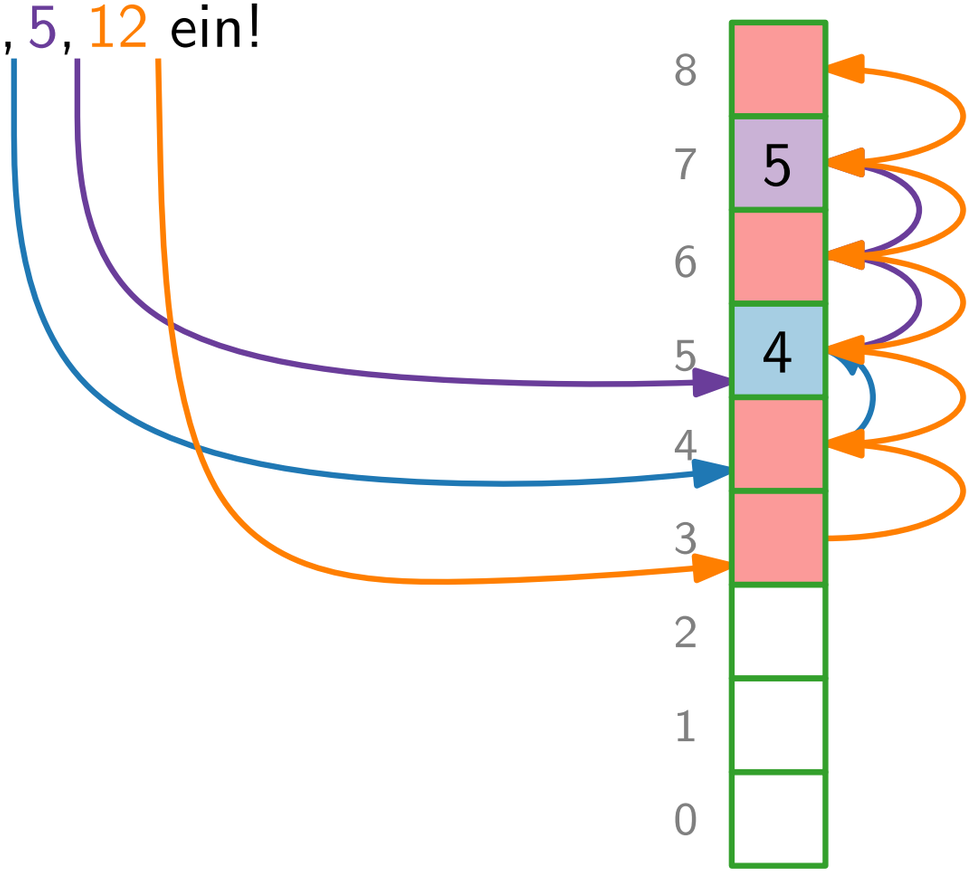


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

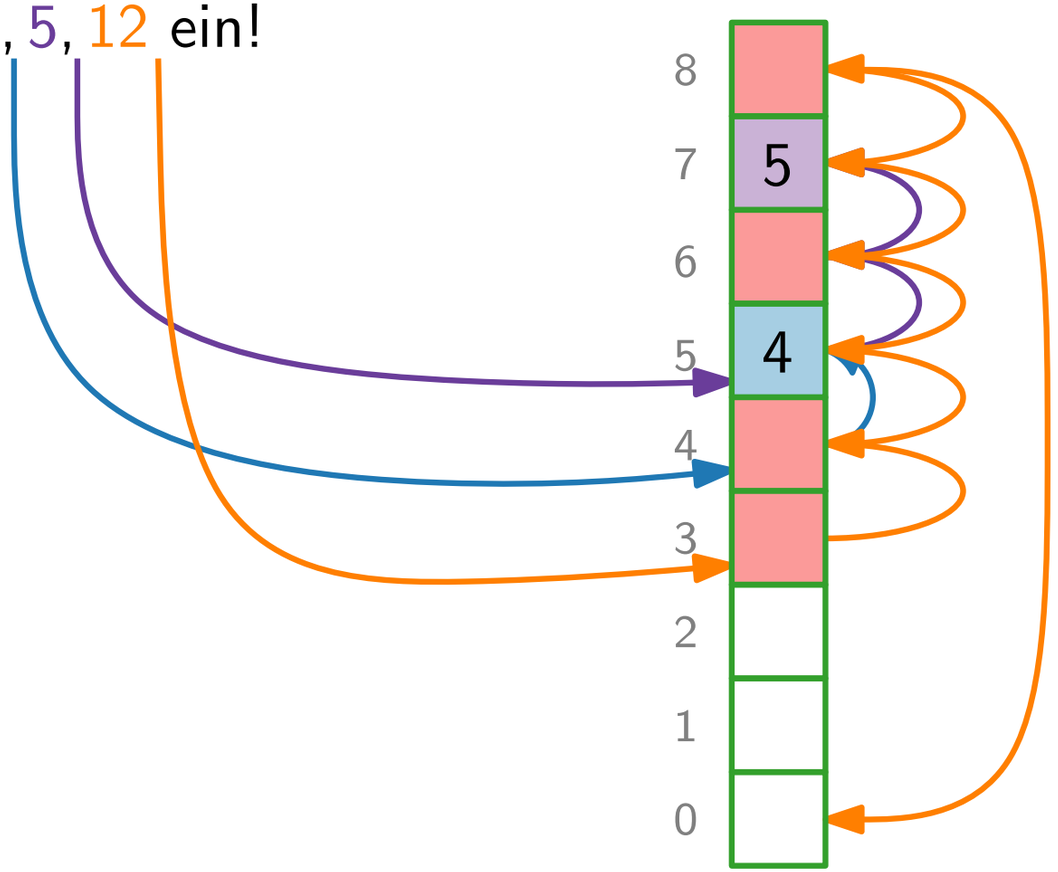


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

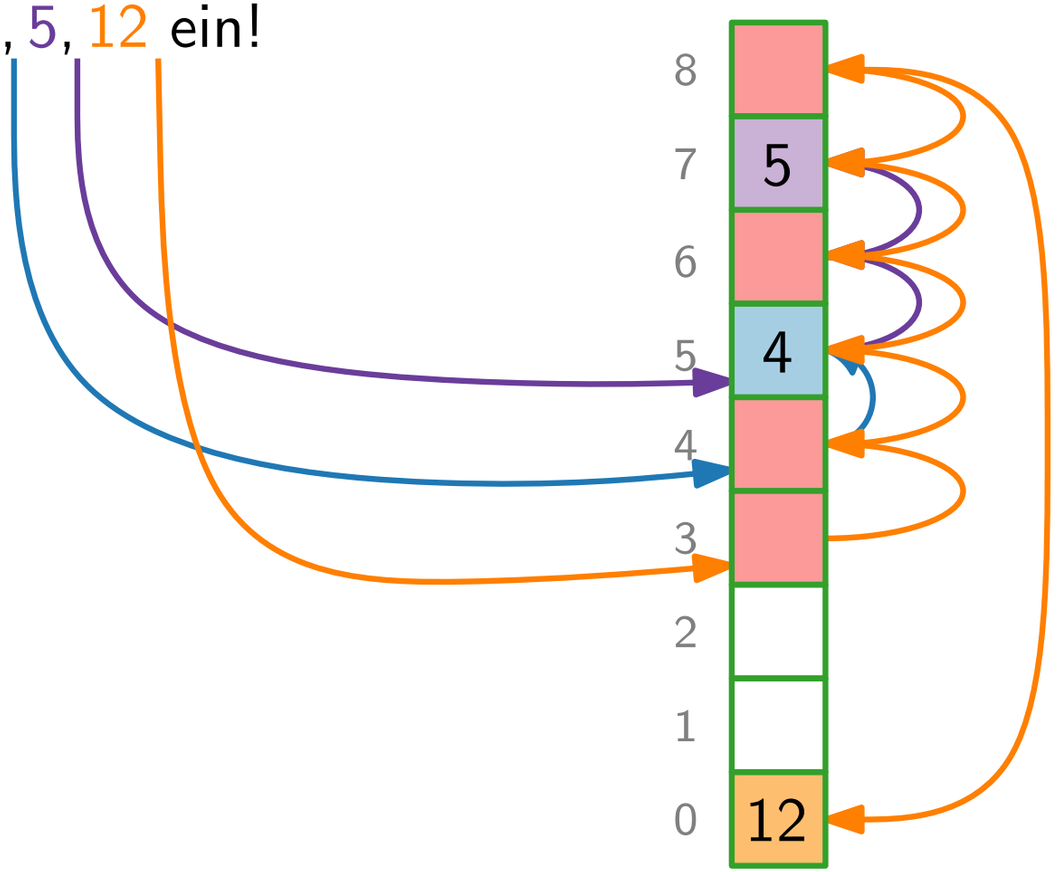


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!

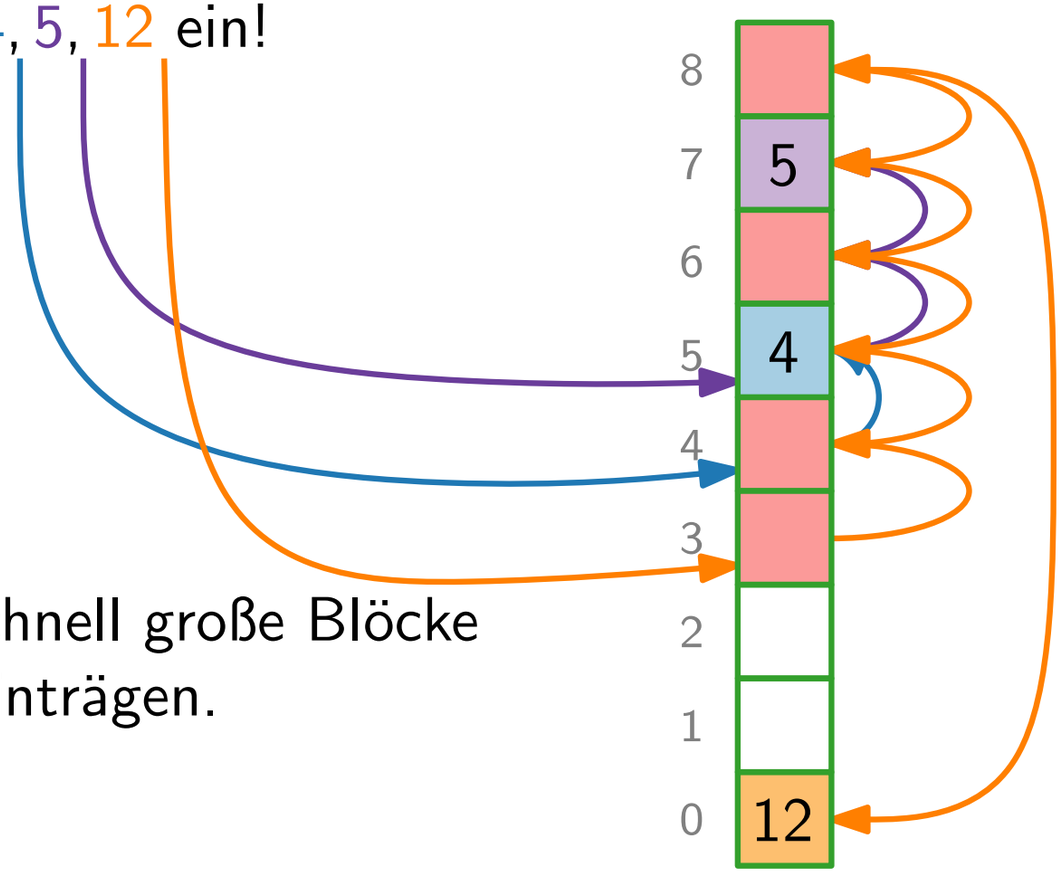


Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!



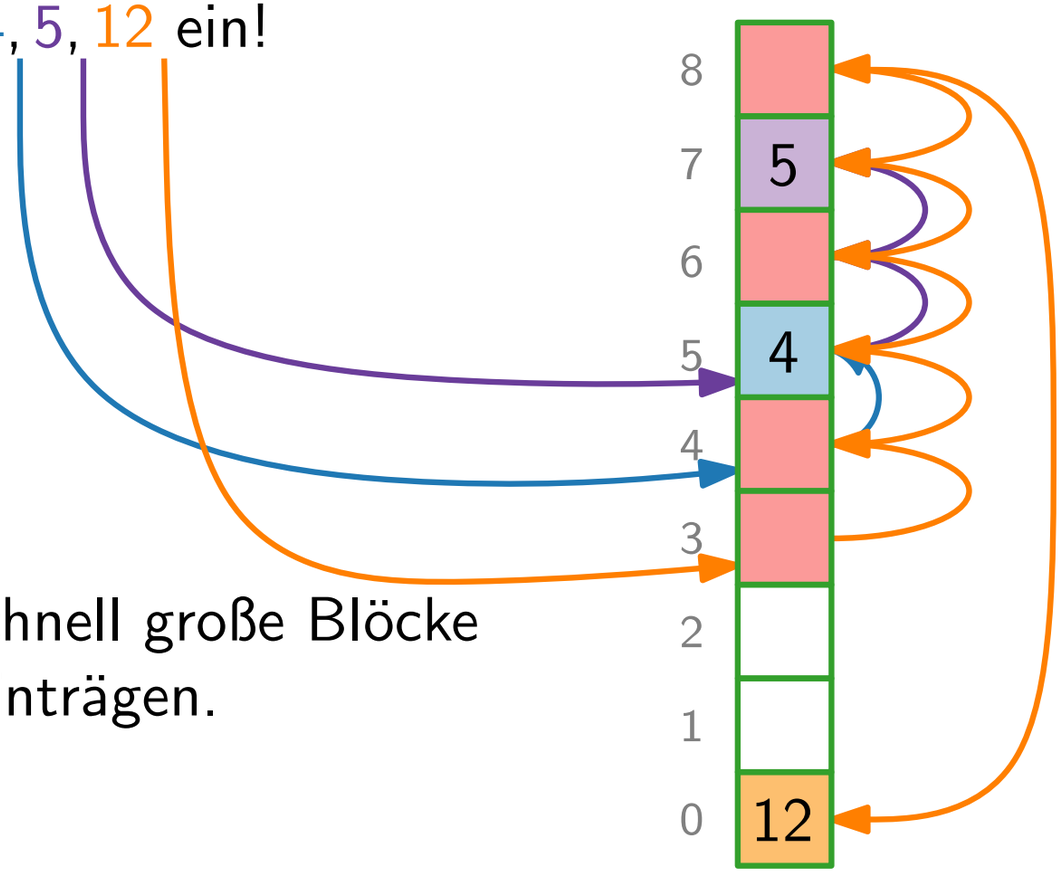
Problem: Es bilden sich schnell große Blöcke von besetzten Einträgen.

Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!



Problem: Es bilden sich schnell große Blöcke von besetzten Einträgen.

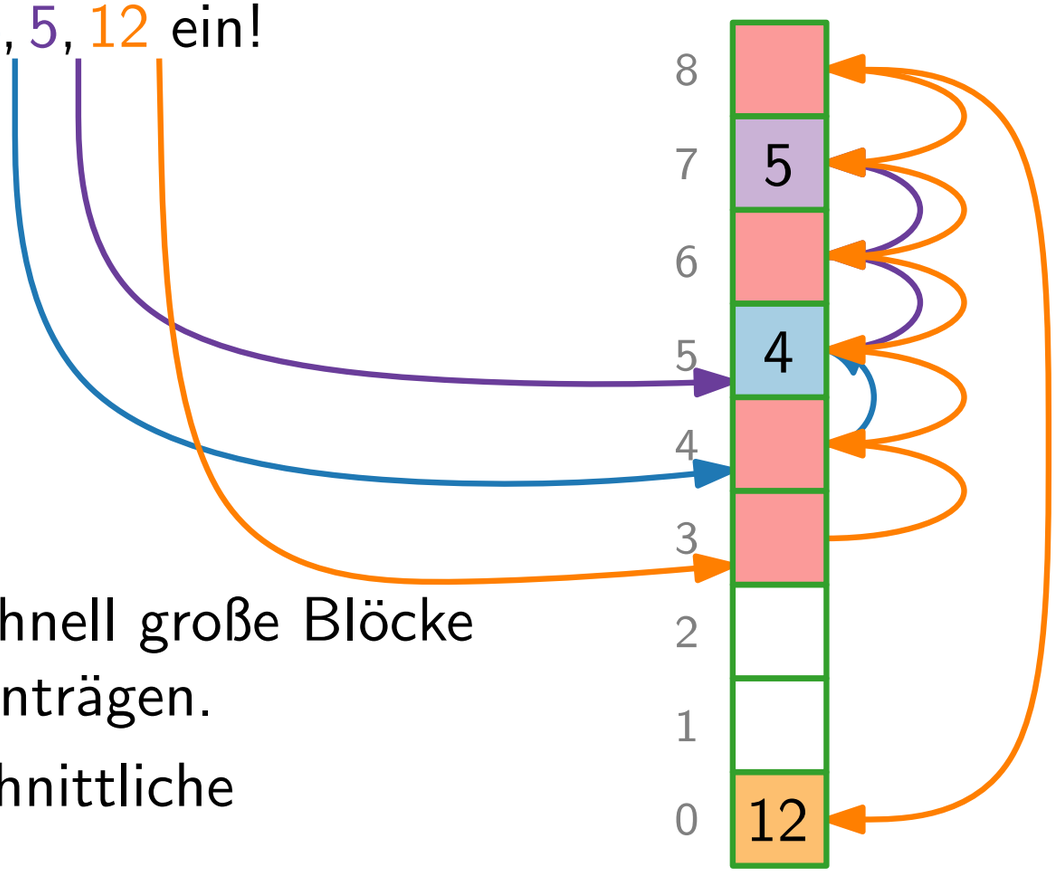
primäres Clustering

Lineares Sondieren

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!



Problem: Es bilden sich schnell große Blöcke von besetzten Einträgen.

primäres Clustering

$\Rightarrow$ hohe durchschnittliche Suchzeit!

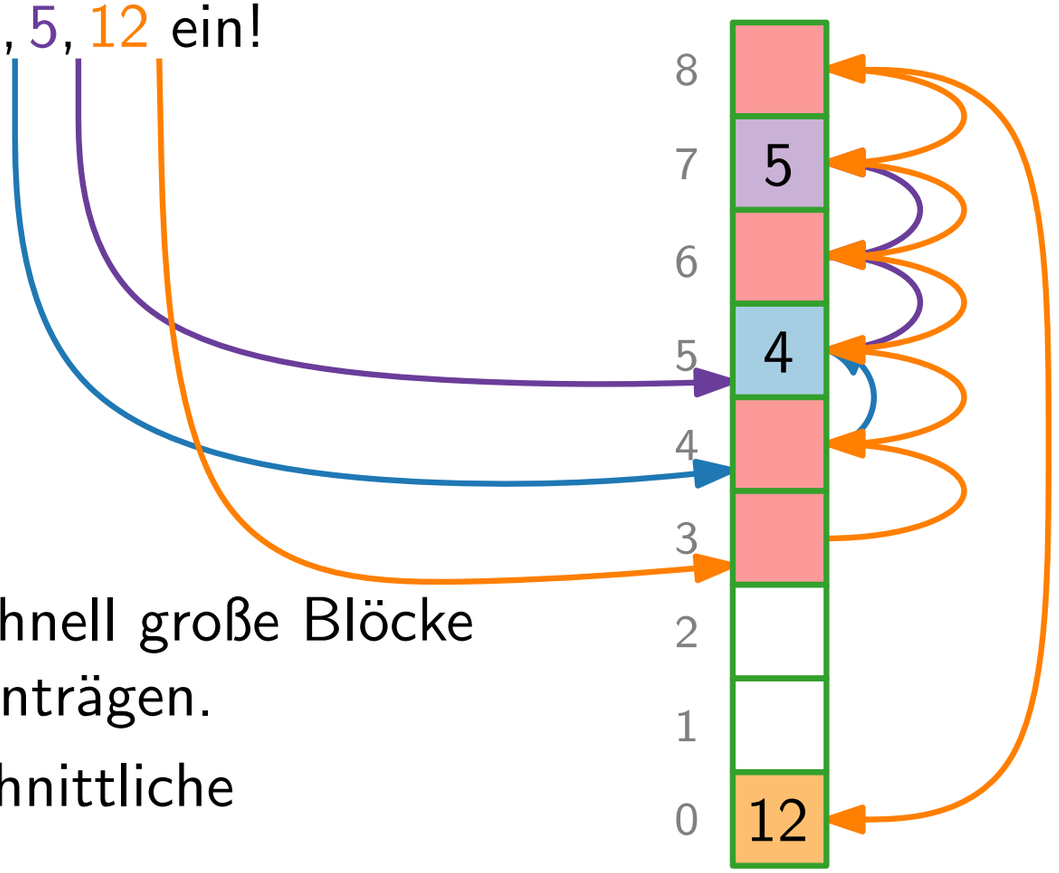
Lineares Sondieren

Demo.
`https://algo.uni-trier.de/demos/hashing.html`

Sondierfolge: $h(k, i) = (h_0(k) + i) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$

Füge Schlüssel 4, 5, 12 ein!



Problem: Es bilden sich schnell große Blöcke von besetzten Einträgen.

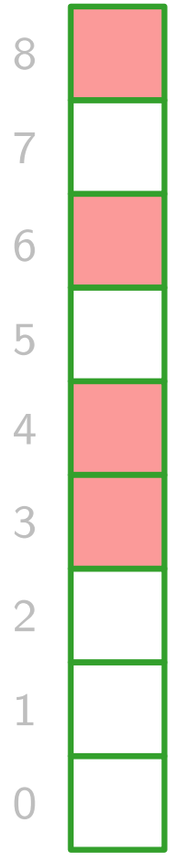
primäres Clustering

$\Rightarrow$ hohe durchschnittliche Suchzeit!

Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

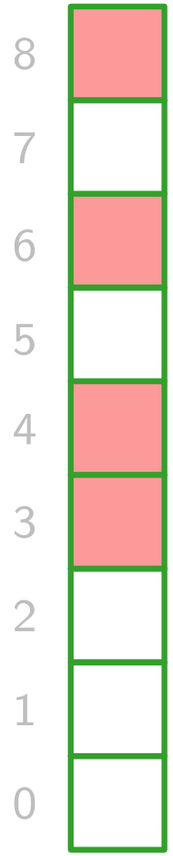
Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$



Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

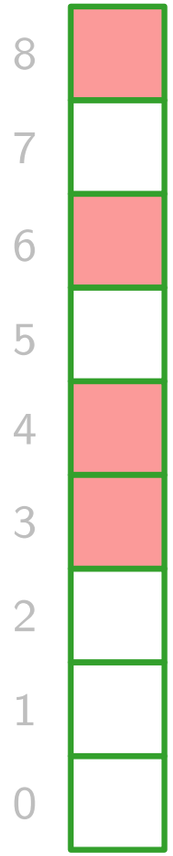


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

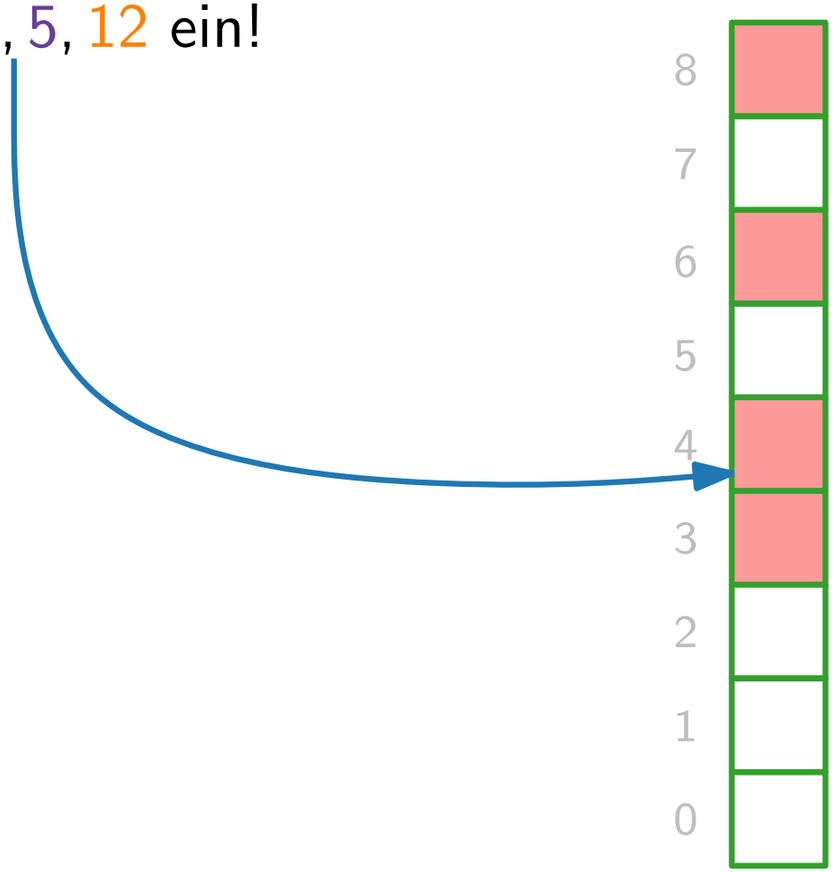


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

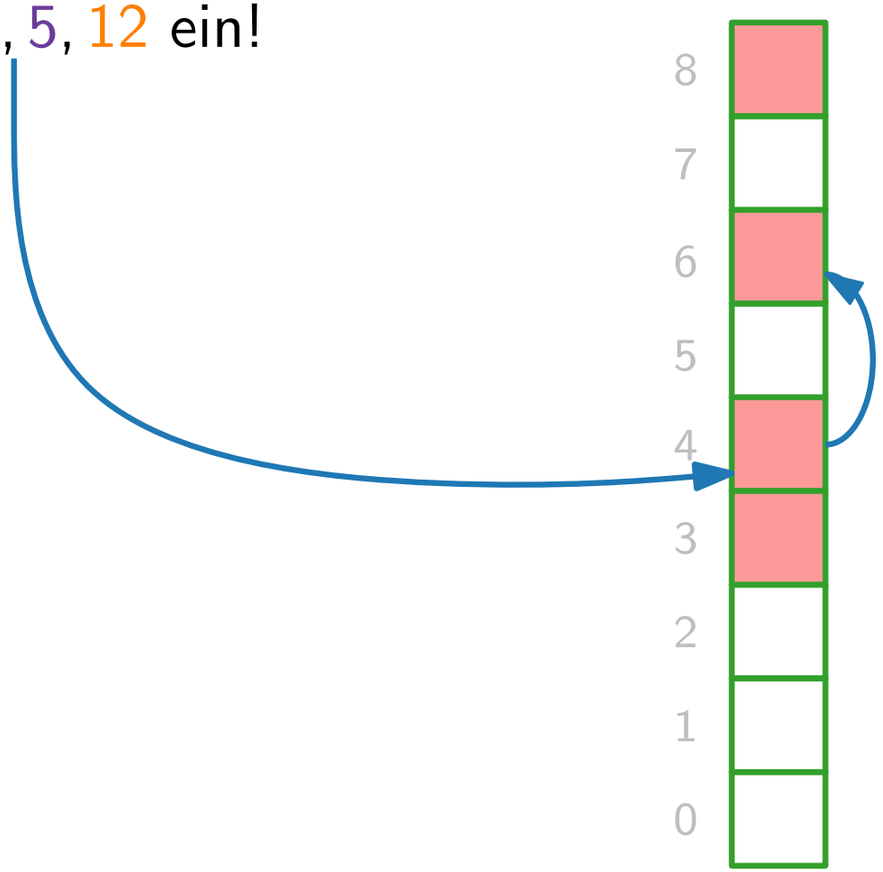


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

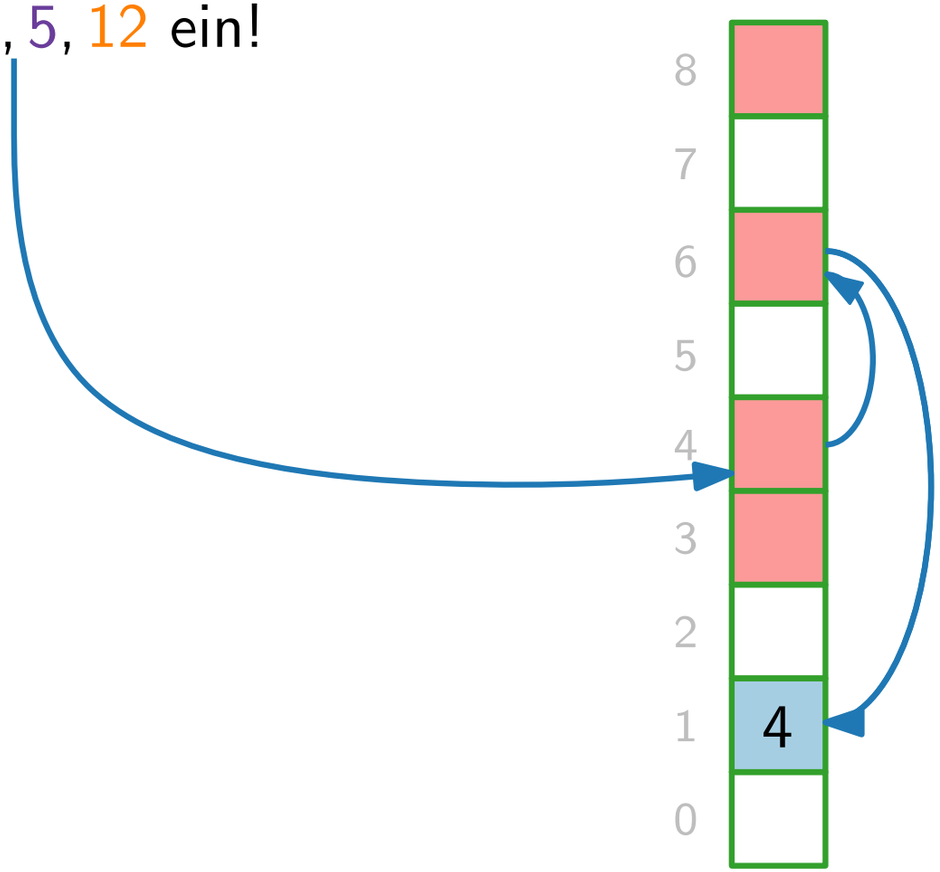


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

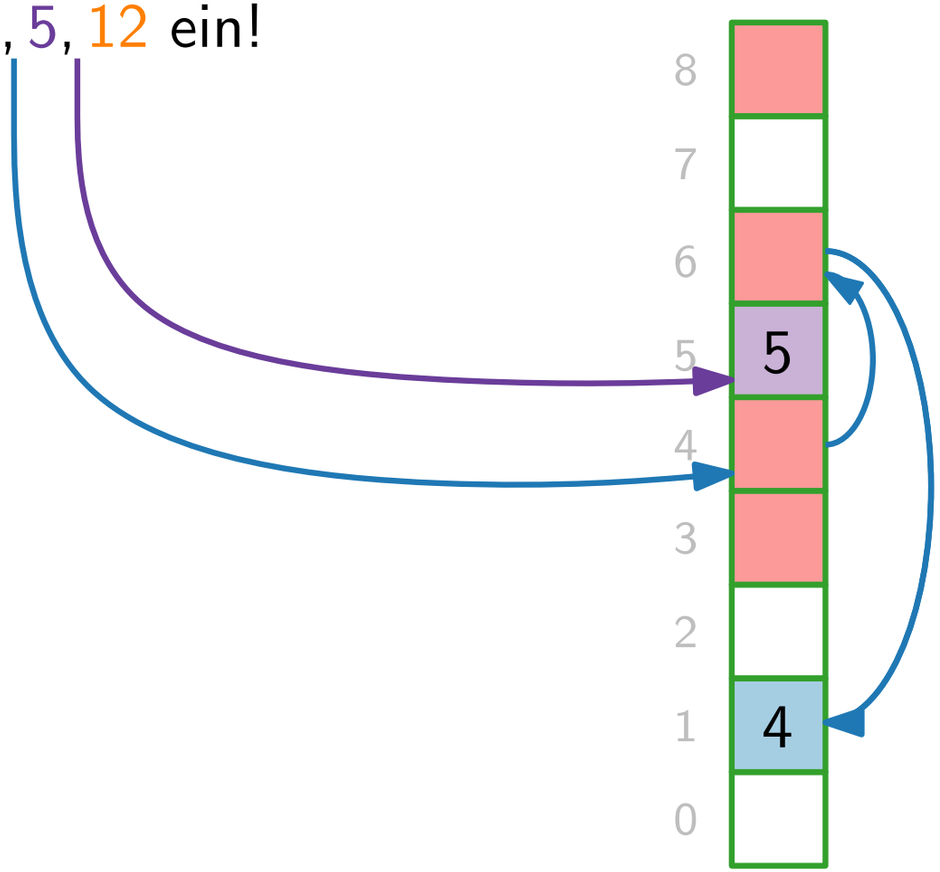


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

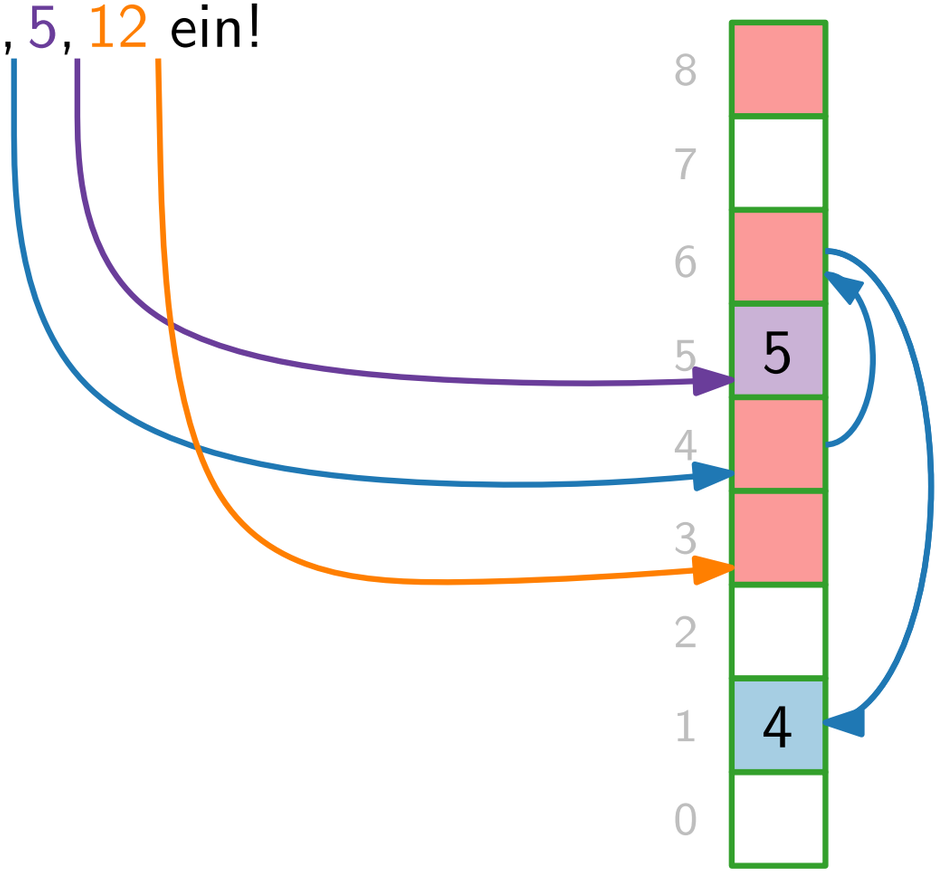


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

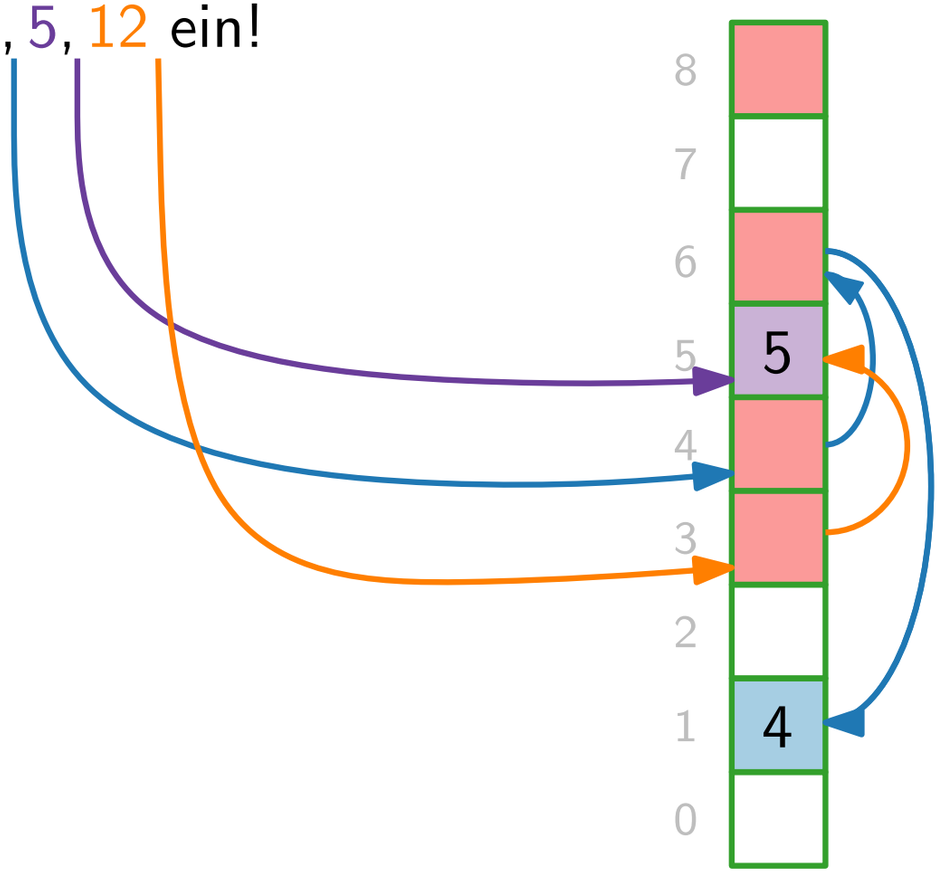


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

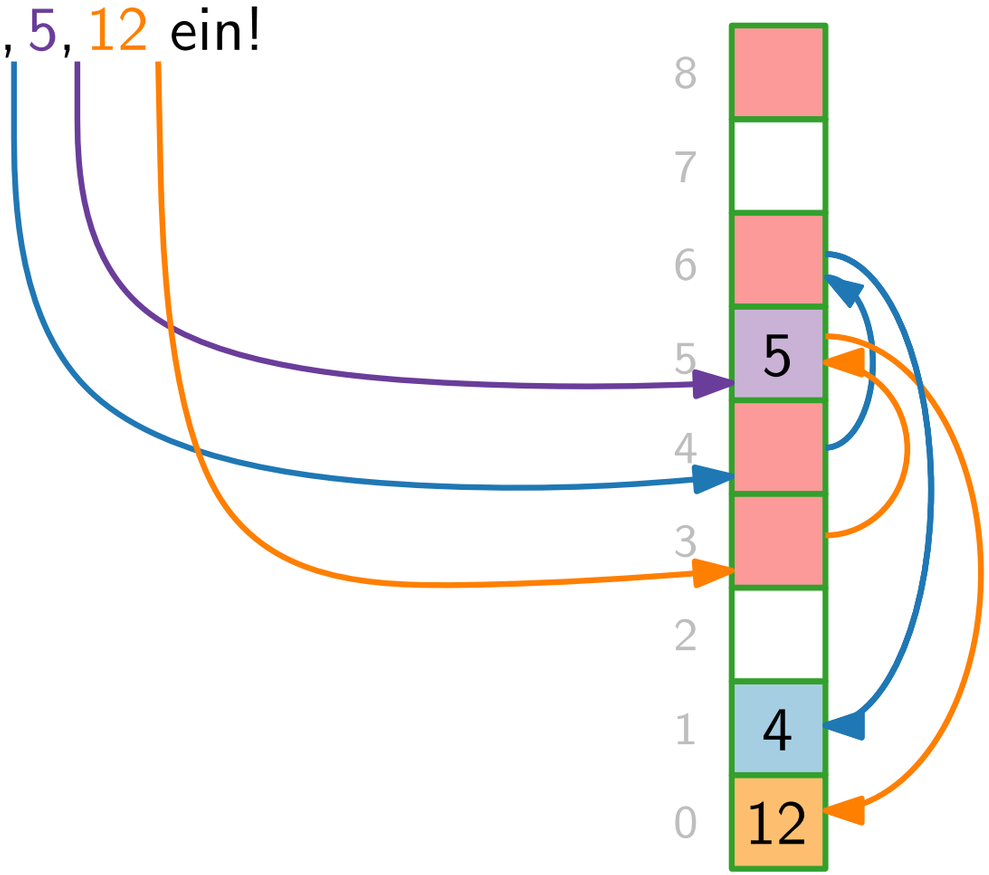


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!



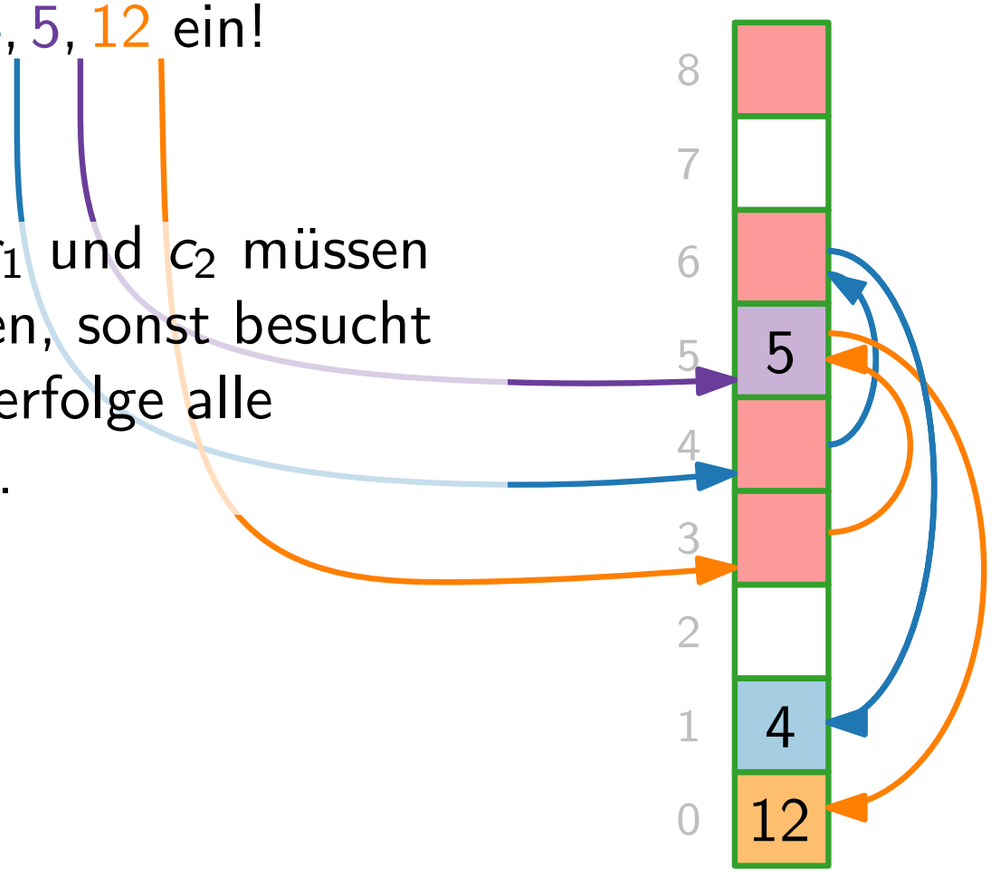
Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!

Problem: Die Größen m, c_1 und c_2 müssen zueinander passen, sonst besucht nicht jede Sondierfolge alle Tabelleneinträge.

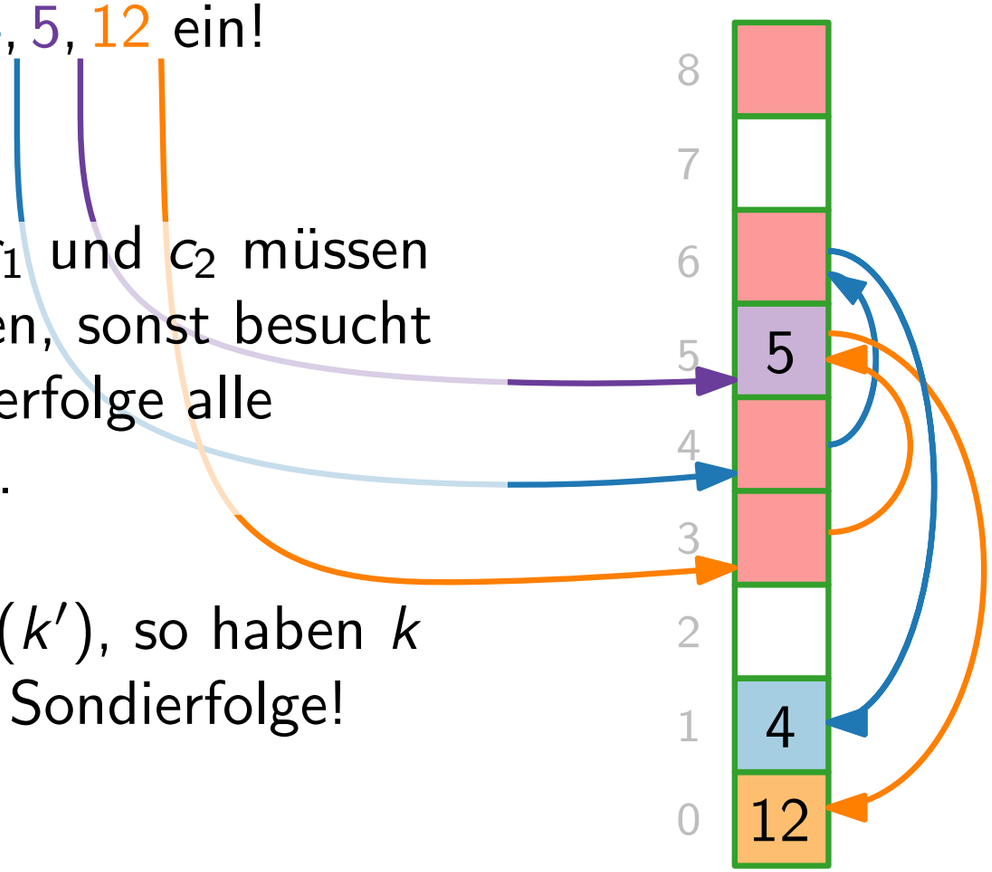


Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!



Problem: Die Größen m, c_1 und c_2 müssen zueinander passen, sonst besucht nicht jede Sondierfolge alle Tabelleneinträge.

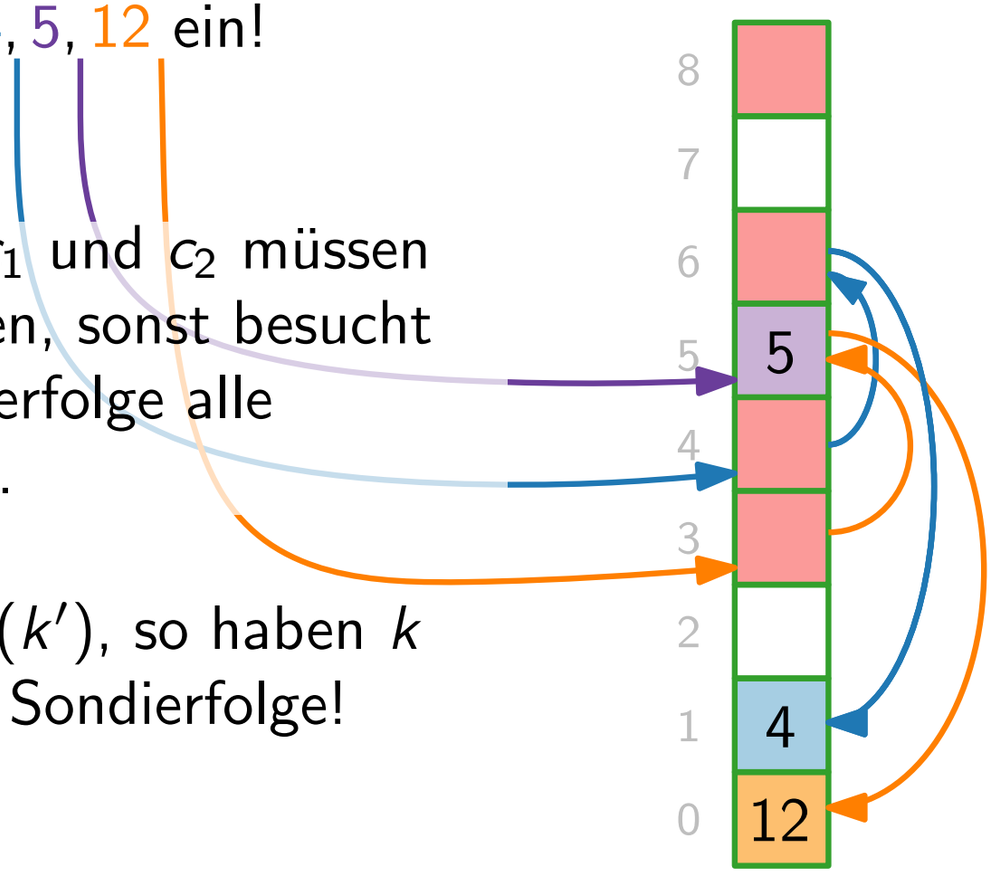
Problem: Falls $h_0(k) = h_0(k')$, so haben k und k' **dieselbe** Sondierfolge!

Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!



Problem: Die Größen m, c_1 und c_2 müssen zueinander passen, sonst besucht nicht jede Sondierfolge alle Tabelleneinträge.

Problem: Falls $h_0(k) = h_0(k')$, so haben k und k' **dieselbe** Sondierfolge!

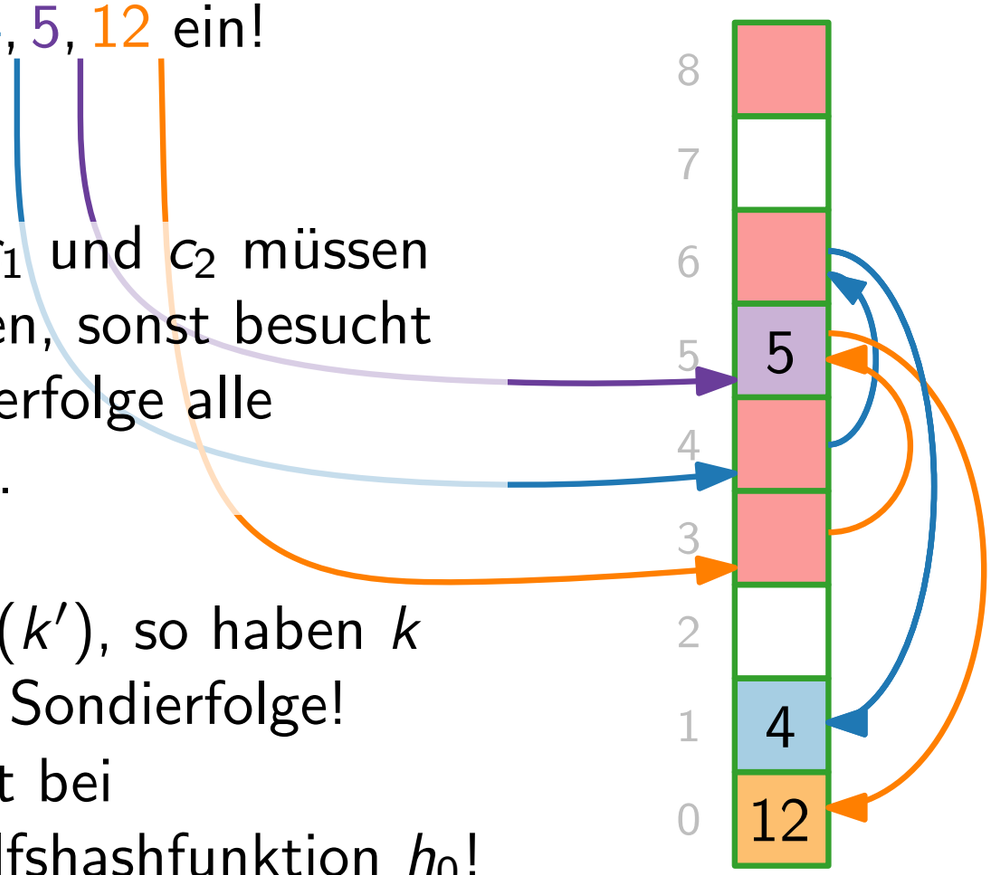
sekundäres Clustering

Quadratisches Sondieren

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!



Problem: Die Größen m, c_1 und c_2 müssen zueinander passen, sonst besucht nicht jede Sondierfolge alle Tabelleneinträge.

Problem: Falls $h_0(k) = h_0(k')$, so haben k und k' **dieselbe** Sondierfolge!

$\Rightarrow$ hohe Suchzeit bei schlechter Hilfshashfunktion h_0!

sekundäres Clustering

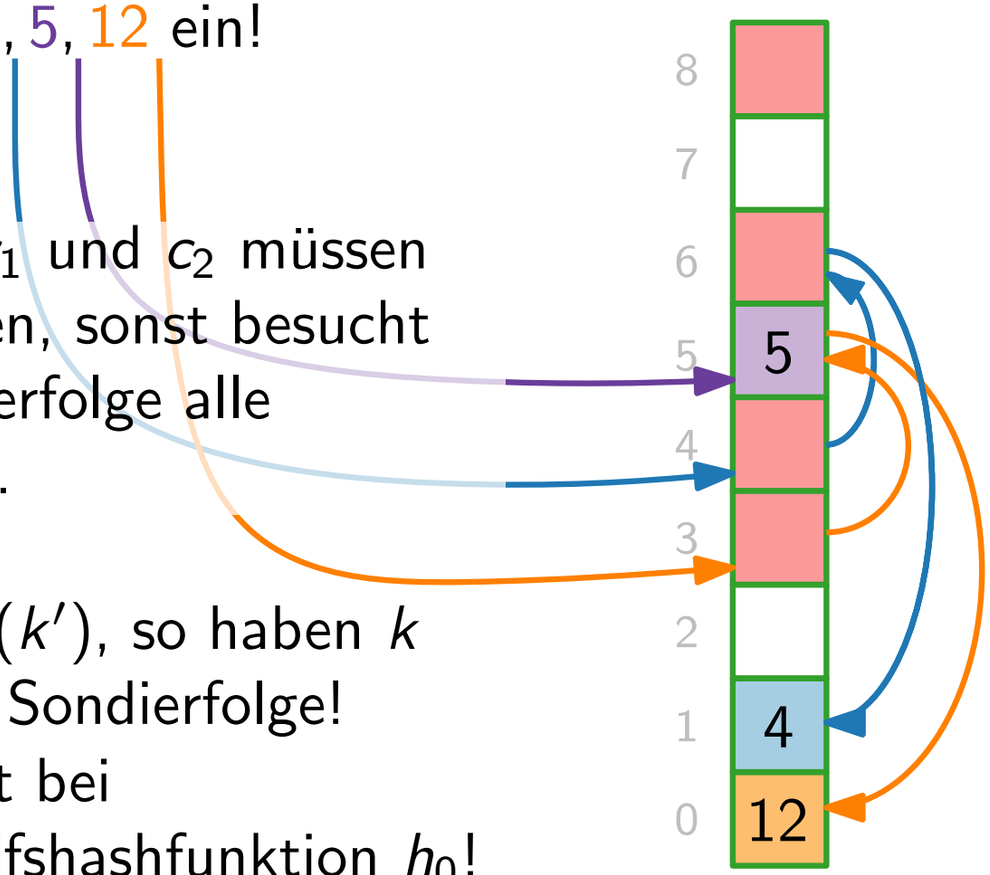
Quadratisches Sondieren

Demo.
`https://algo.uni-trier.de/demos/hashing.html`

Sondierfolge: $h(k,i) = (h_0(k) + c_1 i + c_2 i^2) \bmod m$

Beispiel: $h_0(k) = k \bmod 9$ und $m = 9$ und $c_1 = c_2 = 1$

Füge Schlüssel 4, 5, 12 ein!



Problem: Die Größen m, c_1 und c_2 müssen zueinander passen, sonst besucht nicht jede Sondierfolge alle Tabelleneinträge.

Problem: Falls $h_0(k) = h_0(k')$, so haben k und k' **dieselbe** Sondierfolge!
$\Rightarrow$ hohe Suchzeit bei schlechter Hilfshashfunktion h_0!

sekundäres Clustering

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Antwort:

Doppeltes Hashing

Sondierfolge: $h(k, i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Antwort: $k' = h_1(k)$ und m müssen **teilerfremd** sein

Doppeltes Hashing

Sondierfolge: $h(k,i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Antwort: $k' = h_1(k)$ und m müssen **teilerfremd** sein ,

d.h. $\mathrm{ggT}(k', m) = 1$.

$\mathrm{ggT}(a,b) =_{\text{Def.}} \max\{t : t|a \text{ und } t|b\}$

Doppeltes Hashing

Sondierfolge: $h(k,i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Antwort: $k' = h_1(k)$ und m müssen **teilerfremd** sein ,

d.h. $\mathrm{ggT}(k', m) = 1$.

$\mathrm{ggT}(a,b) =_{\text{Def.}} \max\{t : t|a \text{ und } t|b\}$

Also:

Doppeltes Hashing

Sondierfolge: $h(k,i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Antwort: $k' = h_1(k)$ und m müssen **teilerfremd** sein ,

d.h. $\mathrm{ggT}(k', m) = 1$.

$\mathrm{ggT}(a,b) =_{\text{Def.}} \max\{t : t|a \text{ und } t|b\}$

Also: z.B. $m =$ Zweierpotenz und h_1 immer ungerade.

Doppeltes Hashing

Sondierfolge: $h(k,i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Antwort: $k' = h_1(k)$ und m müssen **teilerfremd** sein ,
d.h. $\mathrm{ggT}(k', m) = 1$.

$\mathrm{ggT}(a,b) =_{\text{Def.}} \max\{t : t|a \text{ und } t|b\}$

Also: z.B. $m =$ Zweierpotenz und h_1 immer ungerade.
oder $m =$ prim und $0 < h_1(k) < m$ für alle k.

Doppeltes Hashing

Demo.
`https://algo.uni-trier.de/demos/hashing.html`

Sondierfolge: $h(k,i) = (h_0(k) + i \cdot h_1(k)) \bmod m$

Vorteile:

- Sondierfolge hängt zweifach vom Schlüssel k ab!
- potentiell m^2 verschiedene Sondierfolgen möglich
  (bei linearem & quadratischem Sondieren nur m.)

Frage: Was muss gelten, damit eine Sondierfolge alle Tabelleneinträge durchläuft?

Antwort: $k' = h_1(k)$ und m müssen **teilerfremd** sein ,
d.h. $\mathrm{ggT}(k', m) = 1$.

$\mathrm{ggT}(a,b) =_{\text{Def.}} \max\{t : t|a \text{ und } t|b\}$

Also: z.B. $m =$ Zweierpotenz und h_1 immer ungerade.
oder $m =$ prim und $0 < h_1(k) < m$ für alle k.

Uniformes Hashing

Uniformes Hashing

kein neues Hashverfahren, sondern eine (idealisierte) Annahme...

Annahme: Die Sondierfolge jedes Schlüssels ist gleich wahrscheinlich eine der $m!$ Permutationen von $\langle 0, 1, \ldots, m-1 \rangle$.

Uniformes Hashing

kein neues Hashverfahren, sondern eine (idealisierte) Annahme...

Annahme: Die Sondierfolge jedes Schlüssels ist gleich wahrscheinlich eine der $m!$ Permutationen von $\langle 0, 1, \ldots, m-1 \rangle$.

Satz. Unter der Annahme von uniformem Hashing ist die erwartete Anzahl der versuchten Tabellenzugriffe bei offener Adressierung und

Uniformes Hashing

kein neues Hashverfahren, sondern eine (idealisierte) Annahme...

Annahme: Die Sondierfolge jedes Schlüssels ist gleich wahrscheinlich eine der $m!$ Permutationen von $\langle 0, 1, \ldots, m-1 \rangle$.

Satz. Unter der Annahme von uniformem Hashing ist die erwartete Anzahl der versuchten Tabellenzugriffe bei offener Adressierung und

- erfolgloser Suche $\leq \frac{1}{1-\alpha}$

Uniformes Hashing

kein neues Hashverfahren, sondern eine (idealisierte) Annahme...

Annahme: Die Sondierfolge jedes Schlüssels ist gleich wahrscheinlich eine der $m!$ Permutationen von $\langle 0, 1, \ldots, m-1 \rangle$.

Satz. Unter der Annahme von uniformem Hashing ist die erwartete Anzahl der versuchten Tabellenzugriffe bei offener Adressierung und

- erfolgloser Suche $\leq \dfrac{1}{1-\alpha}$
- erfolgreicher Suche $\leq \dfrac{1}{\alpha} \ln \dfrac{1}{1-\alpha}$

Uniformes Hashing

kein neues Hashverfahren, sondern eine (idealisierte) Annahme...

Annahme: Die Sondierfolge jedes Schlüssels ist gleich wahrscheinlich eine der $m!$ Permutationen von $\langle 0, 1, \ldots, m-1 \rangle$.

Satz. Unter der Annahme von uniformem Hashing ist die erwartete Anzahl der versuchten Tabellenzugriffe bei offener Adressierung und

- erfolgloser Suche $\leq \frac{1}{1-\alpha}$
- erfolgreicher Suche $\leq \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

d.h. Suche dauert erwartet $\mathcal{O}(1)$ Zeit, falls α konst.

Uniformes Hashing

kein neues Hashverfahren, sondern eine (idealisierte) Annahme...

Annahme: Die Sondierfolge jedes Schlüssels ist gleich wahrscheinlich eine der $m!$ Permutationen von $\langle 0, 1, \ldots, m-1 \rangle$.

Satz. Unter der Annahme von uniformem Hashing ist die erwartete Anzahl der versuchten Tabellenzugriffe bei offener Adressierung und

- erfolgloser Suche $\leq \frac{1}{1-\alpha}$
- erfolgreicher Suche $\leq \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

d.h. Suche dauert erwartet $\mathcal{O}(1)$ Zeit, falls α konst.

(Beweis siehe [CLRS 11.4])

Zusammenfassung

Zusammenfassung

mit Verkettung

mit offener Adressierung

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

mit offener Adressierung

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

Zusammenfassung

mit Verkettung

- ⊕ funktioniert für $n \in \mathcal{O}(m)$
- ⊕ gute erwartete Suchzeit:

mit offener Adressierung

- ⊖ funktioniert nur für $n \leq m$

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

⊕ gute erwartete Suchzeit:
erfolglos: α $\boxed{= n/m}$

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

⊕ gute erwartete Suchzeit:

erfolglos: α $\boxed{= n/m}$

erfolgreich: $1 + \frac{\alpha}{2}$

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

⊕ gute erwartete Suchzeit:

erfolglos: α $\boxed{= n/m}$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

⊕ gute erwartete Suchzeit:

erfolglos: α $\boxed{= n/m}$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Zusammenfassung

mit Verkettung

- ⊕ funktioniert für $n \in \mathcal{O}(m)$
- ⊕ gute erwartete Suchzeit:
  erfolglos: α $\boxed{= n/m}$
  erfolgreich: $1 + \frac{\alpha}{2}$

  [Modell: einfaches uniformes Hashing]
- ⊖ Listenoperationen langsam

mit offener Adressierung

- ⊖ funktioniert nur für $n \leq m$
- ⊖ langsam, wenn $n \approx m$

Zusammenfassung

mit Verkettung

- ⊕ funktioniert für $n \in \mathcal{O}(m)$
- ⊕ gute erwartete Suchzeit:
  erfolglos: α $\boxed{= n/m}$
  erfolgreich: $1 + \frac{\alpha}{2}$

  [Modell: einfaches uniformes Hashing]
- ⊖ Listenoperationen langsam

mit offener Adressierung

- ⊖ funktioniert nur für $n \leq m$
- ⊖ langsam, wenn $n \approx m$

Sondiermethoden:

Zusammenfassung

mit Verkettung

- ⊕ funktioniert für $n \in \mathcal{O}(m)$
- ⊕ gute erwartete Suchzeit:
  erfolglos: α $\boxed{= n/m}$
  erfolgreich: $1 + \frac{\alpha}{2}$

  [Modell: einfaches uniformes Hashing]
- ⊖ Listenoperationen langsam

mit offener Adressierung

- ⊖ funktioniert nur für $n \leq m$
- ⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

⊕ gute erwartete Suchzeit:

erfolglos: α $\boxed{= n/m}$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

Zusammenfassung

mit Verkettung

- ⊕ funktioniert für $n \in \mathcal{O}(m)$
- ⊕ gute erwartete Suchzeit:
  erfolglos: α $\boxed{= n/m}$
  erfolgreich: $1 + \frac{\alpha}{2}$

  [Modell: einfaches uniformes Hashing]
- ⊖ Listenoperationen langsam

mit offener Adressierung

- ⊖ funktioniert nur für $n \leq m$
- ⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

- ⊕ gute erwartete Suchzeit:
  erfolglos: $\frac{1}{1-\alpha}$

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

⊕ gute erwartete Suchzeit:

erfolglos: α $\boxed{= n/m}$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

erfolglos: $\frac{1}{1-\alpha}$

erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

⊕ gute erwartete Suchzeit:

erfolglos: α $\boxed{= n/m}$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam

mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

erfolglos: $\frac{1}{1-\alpha}$

erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

[Modell: uniformes Hashing]

Zusammenfassung

mit Verkettung

- ⊕ funktioniert für $n \in \mathcal{O}(m)$
- ⊕ gute erwartete Suchzeit:
  erfolglos: α $= n/m$
  erfolgreich: $1 + \frac{\alpha}{2}$

  [Modell: einfaches uniformes Hashing]
- ⊖ Listenoperationen langsam

mit offener Adressierung

- ⊖ funktioniert nur für $n \leq m$
- ⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

- ⊕ gute erwartete Suchzeit:
  erfolglos: $\frac{1}{1-\alpha}$
  erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

  [Modell: uniformes Hashing]

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

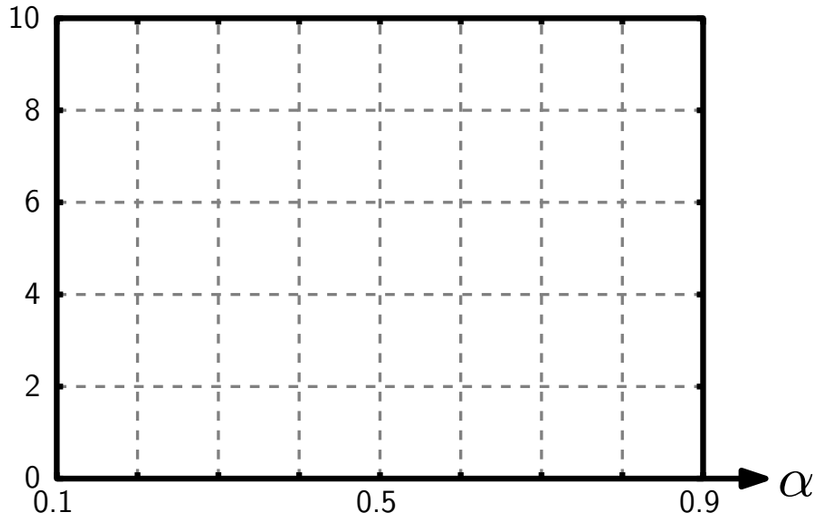
⊕ gute erwartete Suchzeit:

erfolglos: α $= n/m$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam



mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

erfolglos: $\frac{1}{1-\alpha}$

erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

[Modell: uniformes Hashing]

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

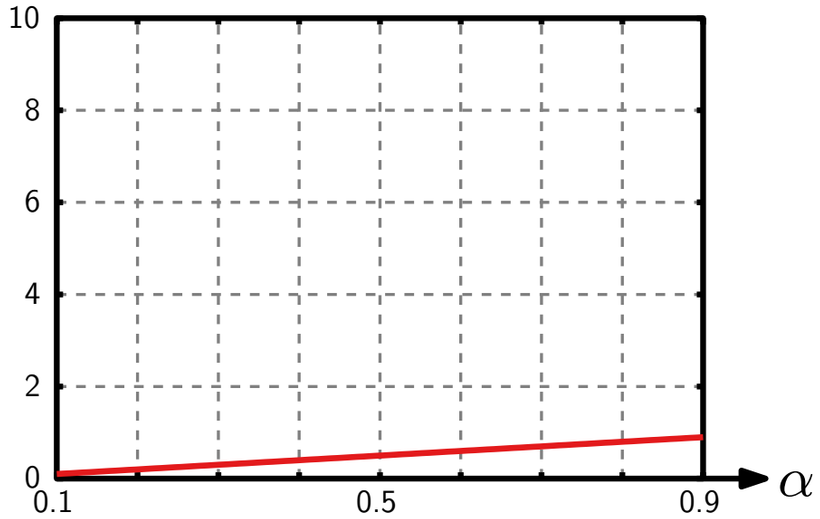
⊕ gute erwartete Suchzeit:

erfolglos: α $= n/m$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam



mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

erfolglos: $\frac{1}{1-\alpha}$

erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

[Modell: uniformes Hashing]

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

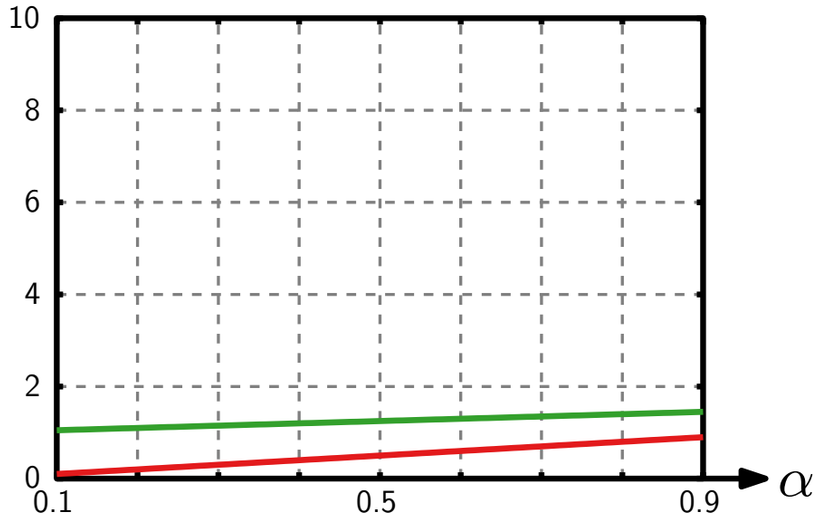
⊕ gute erwartete Suchzeit:

erfolglos: α $= n/m$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam



mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

erfolglos: $\frac{1}{1-\alpha}$

erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

[Modell: uniformes Hashing]

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

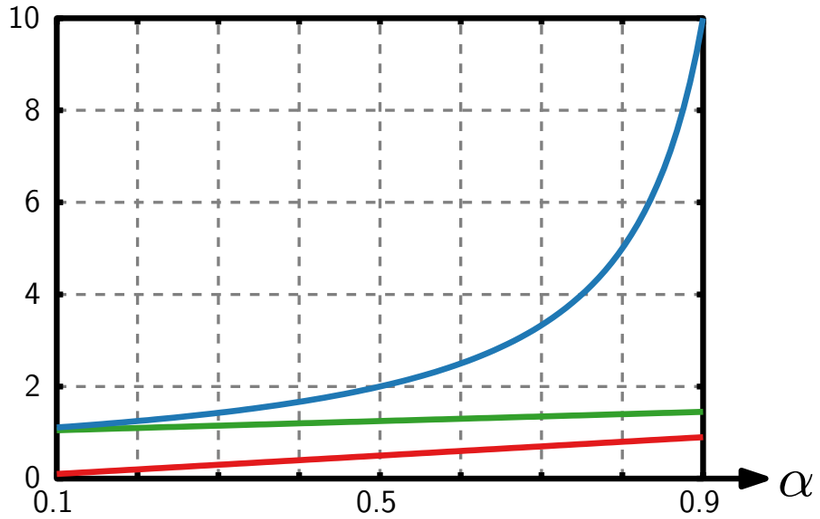
⊕ gute erwartete Suchzeit:

erfolglos: α $= n/m$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam



mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

erfolglos: $\frac{1}{1-\alpha}$

erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

[Modell: uniformes Hashing]

Zusammenfassung

mit Verkettung

⊕ funktioniert für $n \in \mathcal{O}(m)$

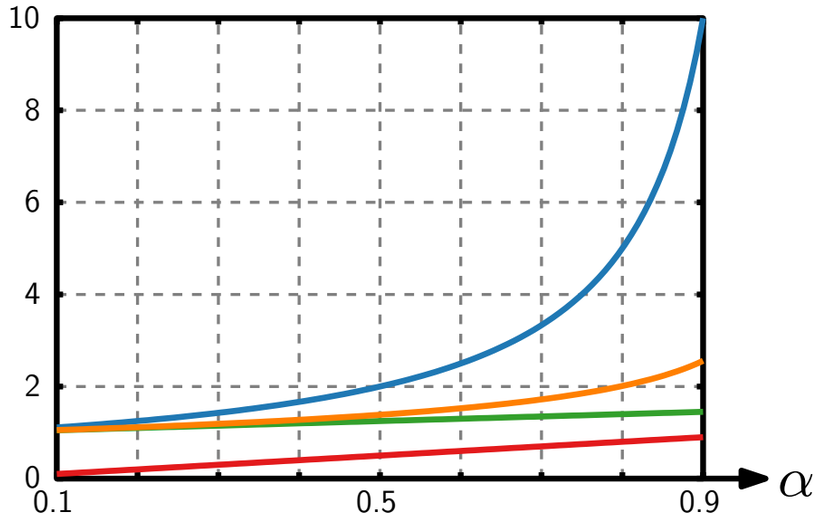
⊕ gute erwartete Suchzeit:

erfolglos: α $= n/m$

erfolgreich: $1 + \frac{\alpha}{2}$

[Modell: einfaches uniformes Hashing]

⊖ Listenoperationen langsam



mit offener Adressierung

⊖ funktioniert nur für $n \leq m$

⊖ langsam, wenn $n \approx m$

Sondiermethoden:

- lineares Sondieren
- quadratisches Sondieren
- doppeltes Hashing

⊕ gute erwartete Suchzeit:

erfolglos: $\frac{1}{1-\alpha}$

erfolgreich: $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

[Modell: uniformes Hashing]