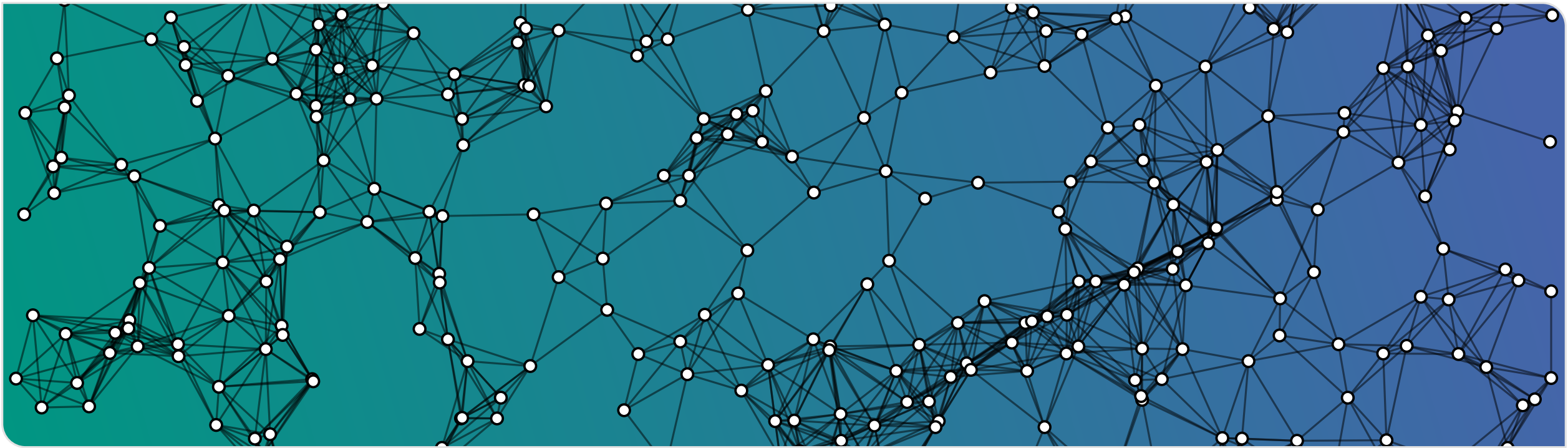
Karlsruher Institut für Technologie

Algorithmen 1

Übung 3
Hashing und Graphen



Klausureinsicht

- Wir wollen die Klausuren in der Einsicht gleichmäßig auf Tische verteilen
- Ein Student der seine Matrikelnummer kennt, soll schnell den richtigen Tisch (Hash*table*) finden

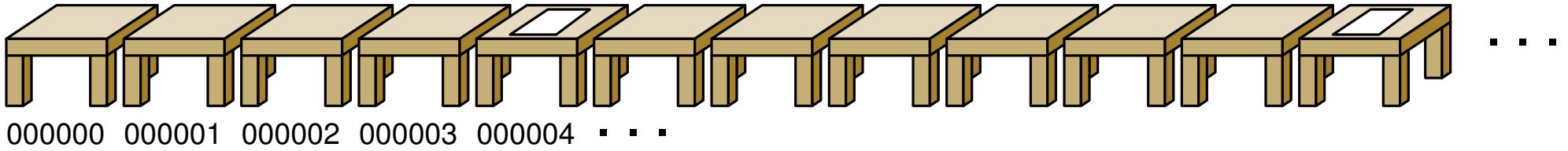
h: Matrikelnummern $\rightarrow$ Tischnummern

Klausureinsicht

- Wir wollen die Klausuren in der Einsicht gleichmäßig auf Tische verteilen
- Ein Student der seine Matrikelnummer kennt, soll schnell den richtigen Tisch (Hash*table*) finden

h: Matrikelnummern $\rightarrow$ Tischnummern

Idee 1
Wenn wir so viele Tische, wie mögliche Matrikelnummer haben, können wir jede Klausur auf ihren eigenen Tisch legen

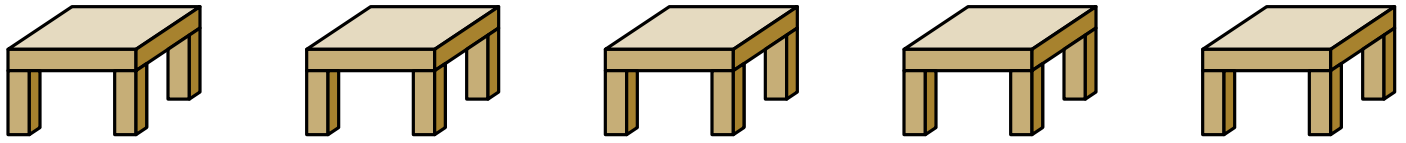


Klausureinsicht

- Wir wollen die Klausuren in der Einsicht gleichmäßig auf Tische verteilen
- Ein Student der seine Matrikelnummer kennt, soll schnell den richtigen Tisch (Hash*table*) finden

h: Matrikelnummern $\rightarrow$ Tischnummern

Idee 2
Wir haben nur fünf Tische aber viel mehr Matrikelnummern
Platziere die Klausuren nach der ersten Ziffer ihrer Matrikelnummer



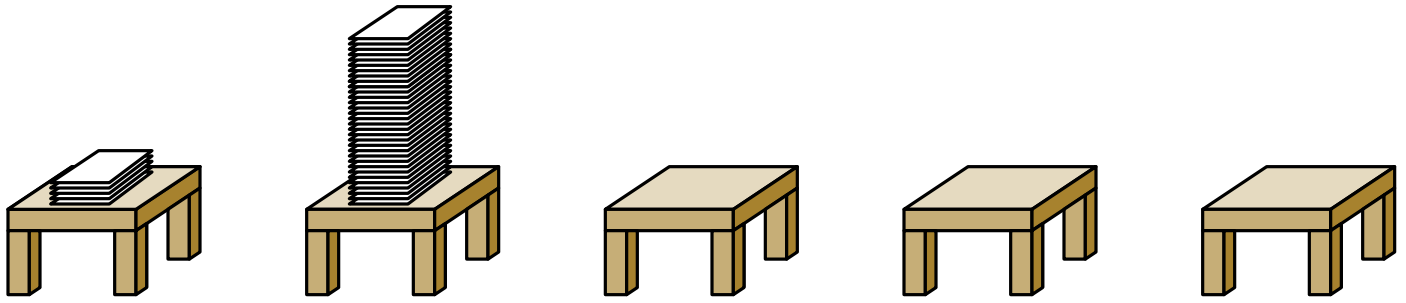
0xxxxx und 1xxxxx | 2xxxxx und 3xxxxx | 4xxxxx und 5xxxxx | 6xxxxx und 7xxxxx | 8xxxxx und 9xxxxx

Klausureinsicht

- Wir wollen die Klausuren in der Einsicht gleichmäßig auf Tische verteilen
- Ein Student der seine Matrikelnummer kennt, soll schnell den richtigen Tisch (Hash*table*) finden

h: Matrikelnummern $\rightarrow$ Tischnummern

Idee 2
Wir haben nur fünf Tische aber viel mehr Matrikelnummern
Platziere die Klausuren nach der ersten Ziffer ihrer Matrikelnummer

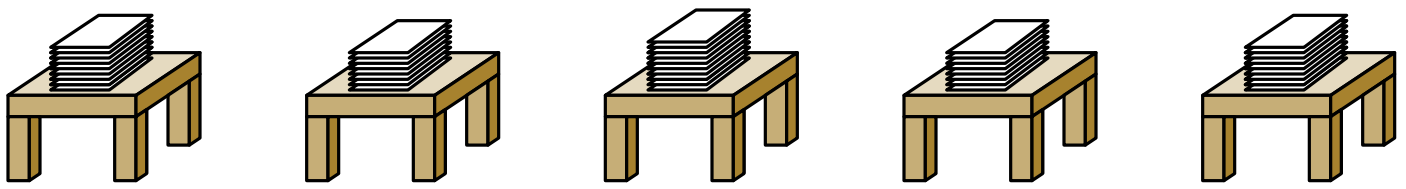


Klausureinsicht

- Wir wollen die Klausuren in der Einsicht gleichmäßig auf Tische verteilen
- Ein Student der seine Matrikelnummer kennt, soll schnell den richtigen Tisch (Hash*table*) finden

h: Matrikelnummern $\rightarrow$ Tischnummern

Idee 3
Platziere die Klausuren nach der **letzten** Ziffer ihrer Matrikelnummer
Diese sind viel *gleichmäßiger verteilt*



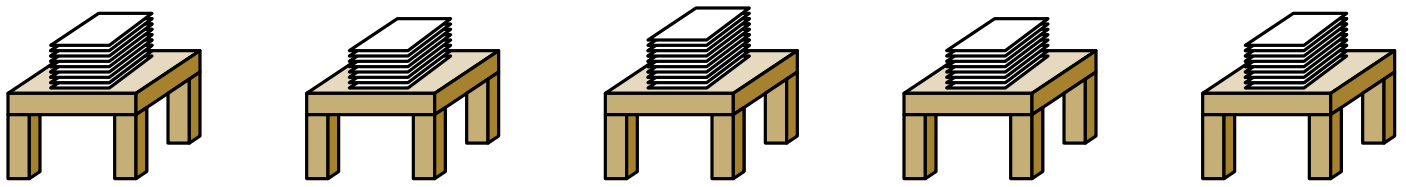
xxxxx0 und xxxxx1 | xxxxx2 und xxxxx3 | xxxxx4 und xxxxx5 | xxxxx6 und xxxxx7 | xxxxx8 und xxxxx9

Klausureinsicht

- Wir wollen die Klausuren in der Einsicht gleichmäßig auf Tische verteilen
- Ein Student der seine Matrikelnummer kennt, soll schnell den richtigen Tisch (Hash*table*) finden

h: Matrikelnummern $\rightarrow$ Tischnummern

Idee 3
Platziere die Klausuren nach der **letzten** Ziffer ihrer Matrikelnummer
Diese sind viel *gleichmäßiger verteilt*



xxxxx0 und xxxxx1 | xxxxx2 und xxxxx3 | xxxxx4 und xxxxx5 | xxxxx6 und xxxxx7 | xxxxx8 und xxxxx9

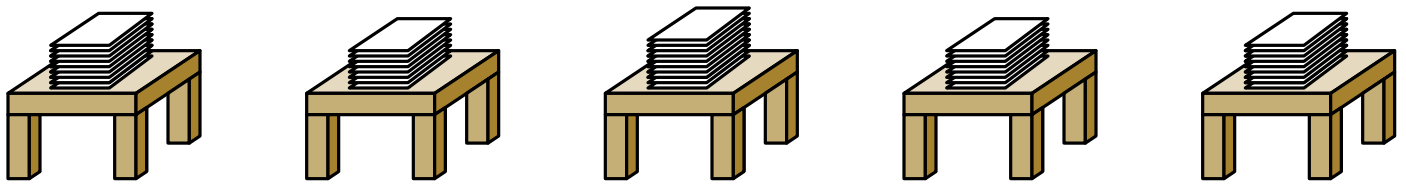
Wenn die Universität die Matrikelnummervergabe ändernt, könnten wir wieder Probleme bekommen

Klausureinsicht

- Wir wollen die Klausuren in der Einsicht gleichmäßig auf Tische verteilen
- Ein Student der seine Matrikelnummer kennt, soll schnell den richtigen Tisch (Hash*table*) finden

h: Matrikelnummern $\rightarrow$ Tischnummern

Idee 3
Platziere die Klausuren nach der **letzten** Ziffer ihrer Matrikelnummer
Diese sind viel *gleichmäßiger verteilt*



Wenn die Universität die Matrikelnummervergabe ändernt, könnten wir wieder Probleme bekommen
$\Rightarrow$ Gibt es ein Verfahren, dass immer funktioniert?

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

- Datenstruktur: **Hashtabelle**

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	
⋮	
7	
8	
9	

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

9

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	
⋮	
7	
8	
9	

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

9

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	
⋮	
7	
8	
9	📖 PickleRick

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

1

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	
⋮	
7	
8	
9	📖 PickleRick

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

1

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	📖 Monkeee
⋮	
7	
8	
9	📖 PickleRick

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

7

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	📖 Monkeee
⋮	
7	
8	
9	📖 PickleRick

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

7

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	📖 Monkeee
⋮	
7	📖 el0nMarsk
8	
9	📖 PickleRick

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

8

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	📖 Monkeee
⋮	
7	📖 el0nMarsk
8	
9	📖 PickleRick

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Mithilfe der Matrikelnummer eines Studenten, soll seine Klausur gefunden werden

MatrikelNr.	Klausur
2035819	📖 PickleRick
2031951	📖 Monkeee
2671947	📖 el0nMarsk
2175809	📖 Stewdent23
2739158	📖 ProfB

8

- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet auf die letzte Ziffer der Matrikelnummer ab

0	
1	📖 Monkeee
⋮	
7	📖 el0nMarsk
8	
9	📖 PickleRick
9	📖 Stewdent23

Hash-Kollision!
$h(x) = h(y)$ obwohl $x \neq y$

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)

→ Gleich mehr dazu

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)

Gleich mehr dazu

Aber bitte nicht alle!

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße

Gleich mehr dazu

Aber bitte nicht alle!

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Simple Uniform Hashing Assumption**
    - Ziehe eine zufällige Hashfunktion

Gleich mehr dazu

Aber bitte nicht alle!

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Simple Uniform Hashing Assumption**
    - Ziehe eine zufällige Hashfunktion
    - Jeder Schlüssel landet in jedem der Buckets mit der gleichen Wahrscheinlichkeit

Gleich mehr dazu

Aber bitte nicht alle!

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Simple Uniform Hashing Assumption**
    - Ziehe eine zufällige Hashfunktion
    - Jeder Schlüssel landet in jedem der Buckets mit der gleichen Wahrscheinlichkeit

Gleich mehr dazu

Aber bitte nicht alle!

Ist die Anzahl eingefügter (Key, Value)-Paare linear in m, landen davon *erwartet* nur konstant viele in einem Bucket.

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Simple Uniform Hashing Assumption**
    - Ziehe eine zufällige Hashfunktion
    - Jeder Schlüssel landet in jedem der Buckets mit der gleichen Wahrscheinlichkeit

Gleich mehr dazu

Aber bitte nicht alle!

Ist die Anzahl eingefügter (Key, Value)-Paare linear in m, landen davon *erwartet* nur konstant viele in einem Bucket.

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array) Aber bitte nicht alle!
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Simple Uniform Hashing Assumption**
    - Ziehe eine zufällige Hashfunktion
    - Jeder Schlüssel landet in jedem der Buckets mit der gleichen Wahrscheinlichkeit
    - Neue Hashfunktion ziehen, wenn Tabelle wächst

Gleich mehr dazu

Ist die Anzahl eingefügter (Key, Value)-Paare linear in m, landen davon *erwartet* nur konstant viele in einem Bucket.

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array) Aber bitte nicht alle!
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Simple Uniform Hashing Assumption**
    - Ziehe eine zufällige Hashfunktion
    - Jeder Schlüssel landet in jedem der Buckets mit der gleichen Wahrscheinlichkeit
    - Neue Hashfunktion ziehen, wenn Tabelle wächst

Gleich mehr dazu

Ist die Anzahl eingefügter (Key, Value)-Paare linear in m, landen davon *erwartet* nur konstant viele in einem Bucket.

Für den Erwartungswert ziehen wir keine zufälligen Keys, sondern zufällige Hashfunktionen

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary 😈: Kann Instanz bauen, wo alle $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Methoden, um mit Kollisionen umzugehen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array) Aber bitte nicht alle!
  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Simple Uniform Hashing Assumption**
    - Ziehe eine zufällige Hashfunktion
    - Jeder Schlüssel landet in jedem der Buckets mit der gleichen Wahrscheinlichkeit
    - Neue Hashfunktion ziehen, wenn Tabelle wächst

Gleich mehr dazu

Ist die Anzahl eingefügter (Key, Value)-Paare linear in m, landen davon *erwartet* nur konstant viele in einem Bucket.

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Für den Erwartungswert ziehen wir keine zufälligen Keys, sondern zufällige Hashfunktionen
$\Rightarrow$ Wir brauchen keine Annahmen über die Key-Verteilung

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Notation

- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen.

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen. (bezüglich Folge ausgeführter Operationen)

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen. (bezüglich Folge ausgeführter Operationen)

Key Takeaways

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen. (bezüglich Folge ausgeführter Operationen)

Key Takeaways

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen. (bezüglich Folge ausgeführter Operationen)

Key Takeaways

- **ERWARTET**

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen. (bezüglich Folge ausgeführter Operationen)

Key Takeaways

- **ERWARTET**
- funktioniert in der Praxis richtig gut

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen. (bezüglich Folge ausgeführter Operationen)

Key Takeaways

- **ERWARTET**
- funktioniert in der Praxis richtig gut
- selbst eine Hashfunktion zu entwerfen ist typischerweise keine gute Idee...

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Open Addressing

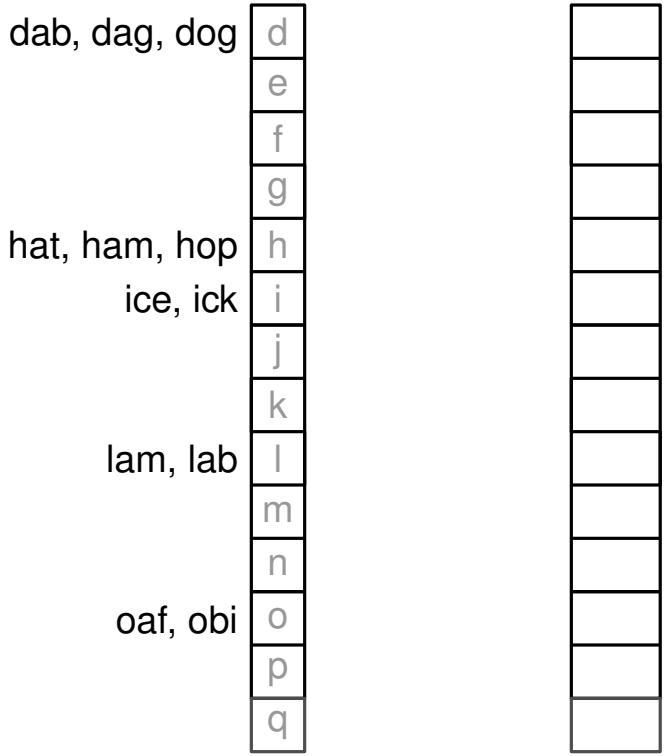
- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

- Nutze ersten Buchstaben als Hashfunktion
- keys = values

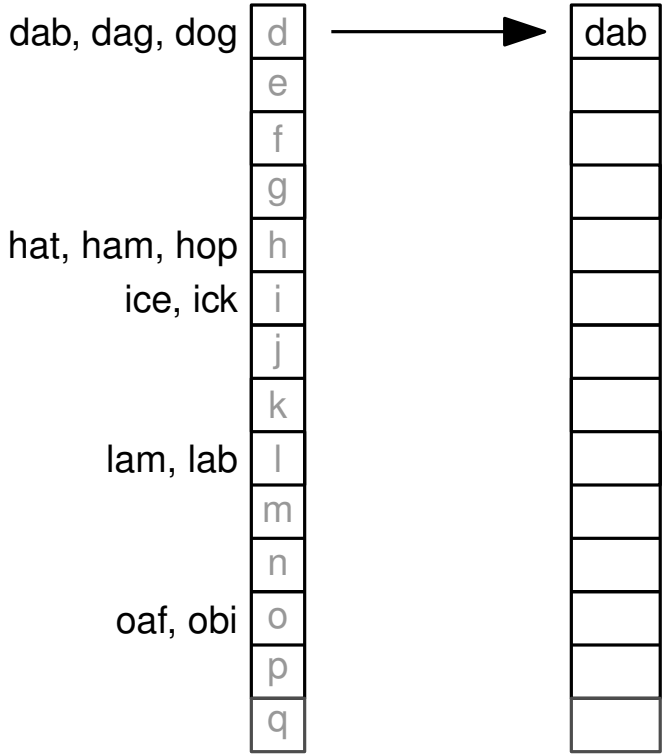


Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

- Nutze ersten Buchstaben als Hashfunktion
- keys = values

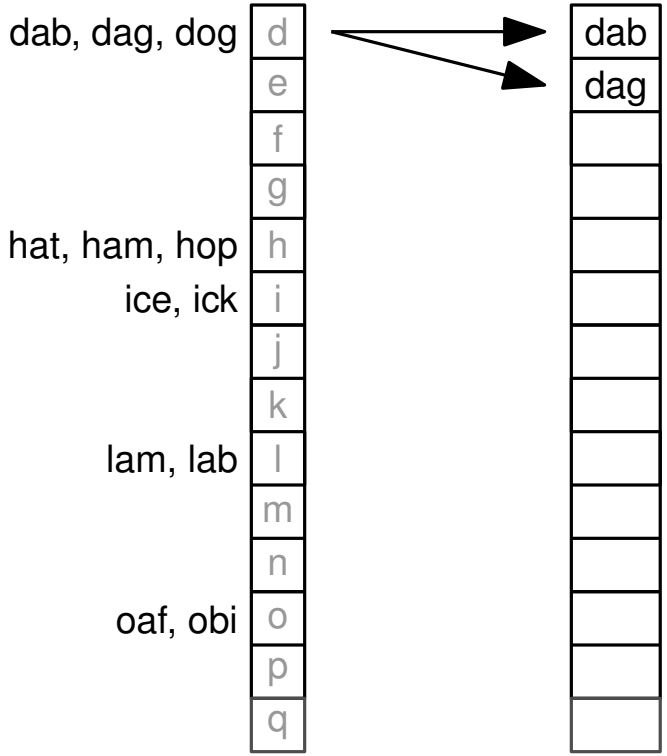


Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

- Nutze ersten Buchstaben als Hashfunktion
- keys = values

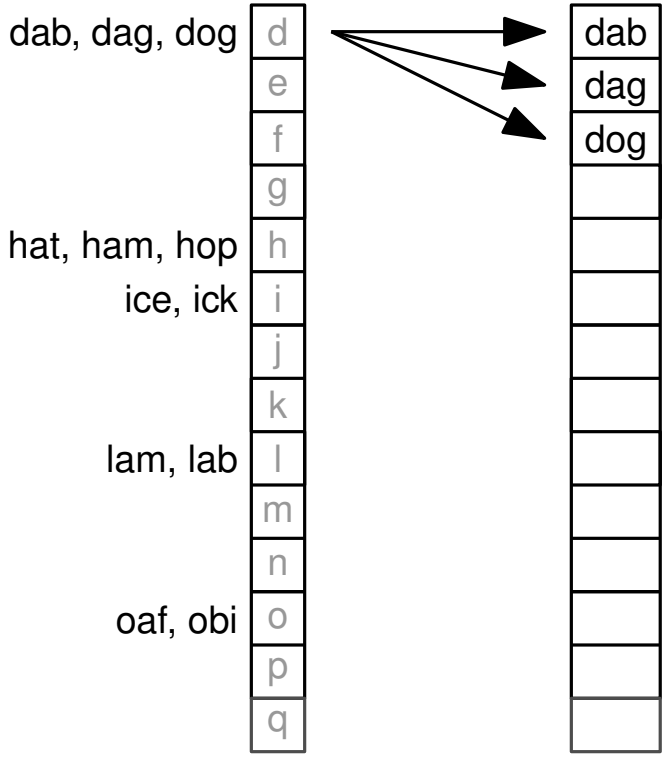


Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

- Nutze ersten Buchstaben als Hashfunktion
- keys = values

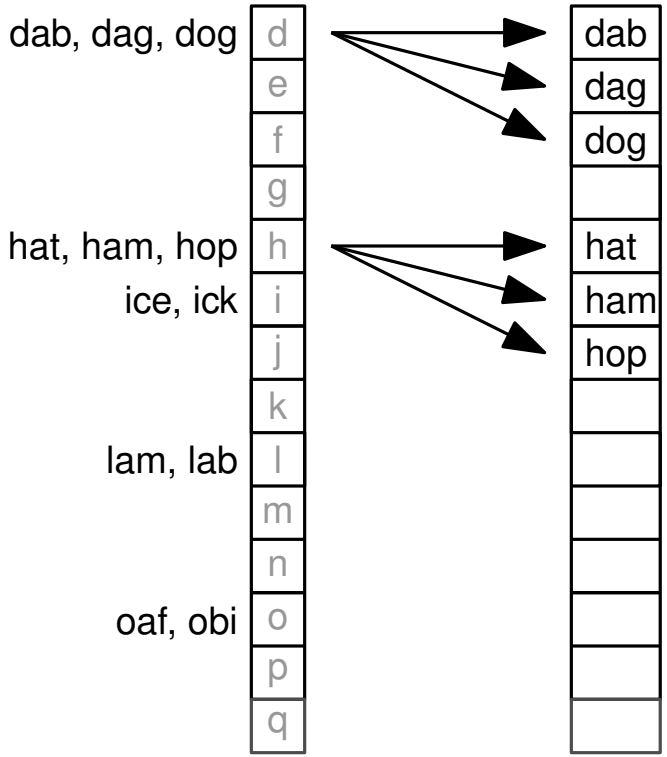


Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

- Nutze ersten Buchstaben als Hashfunktion
- keys = values

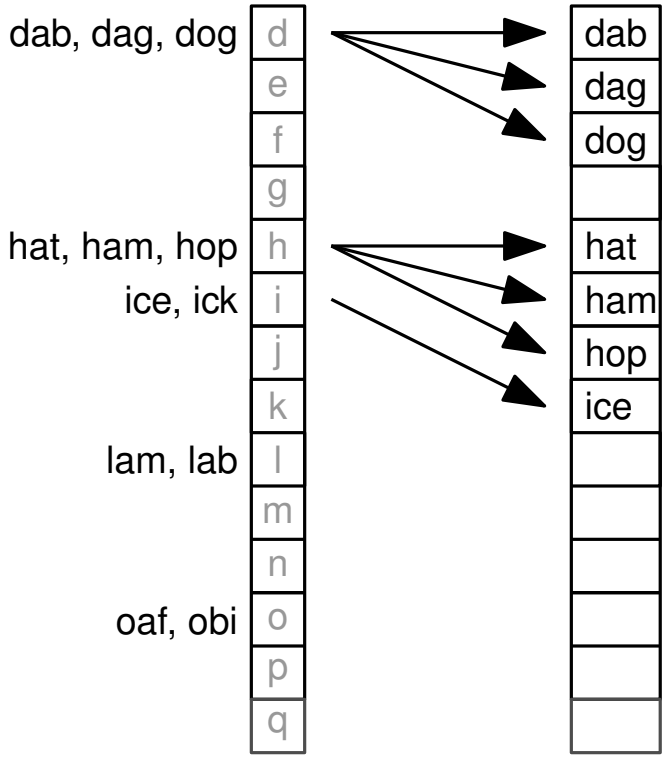


Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

- Nutze ersten Buchstaben als Hashfunktion
- keys = values

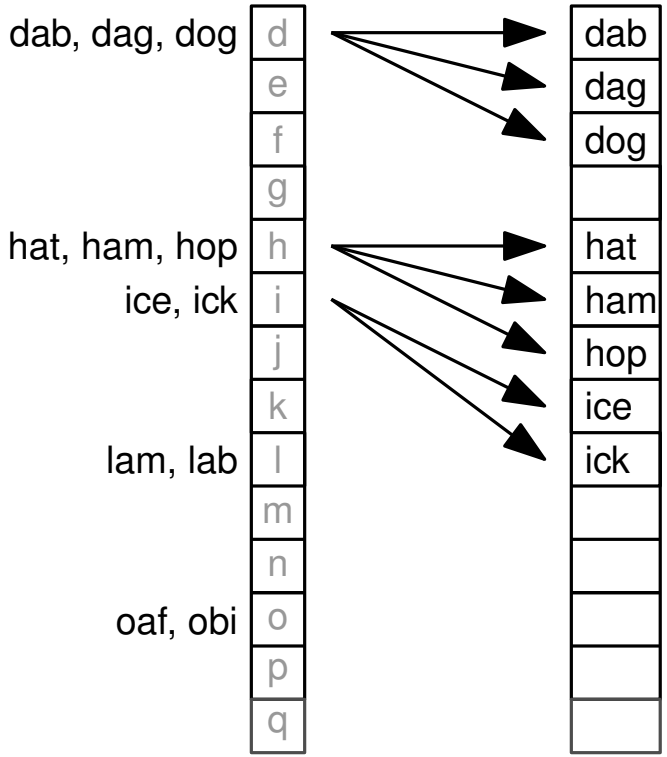


Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

- Nutze ersten Buchstaben als Hashfunktion
- keys = values

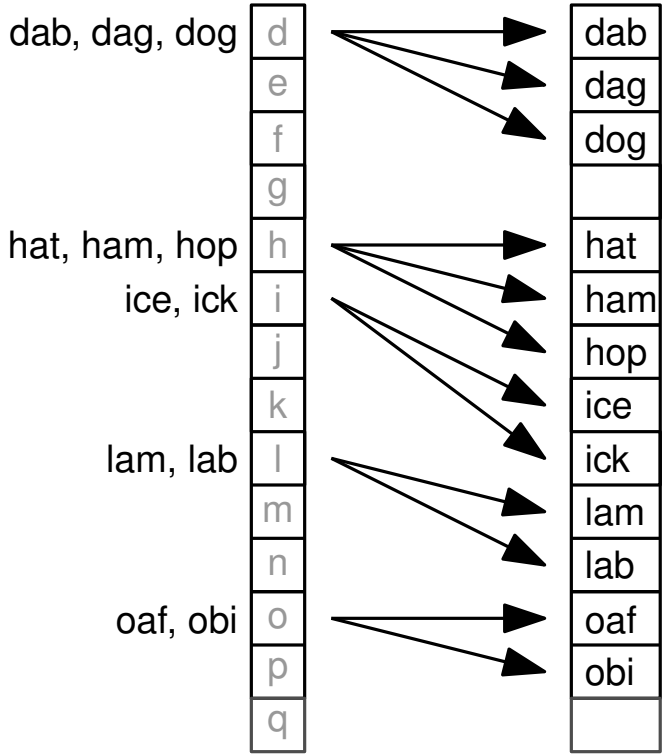


Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:

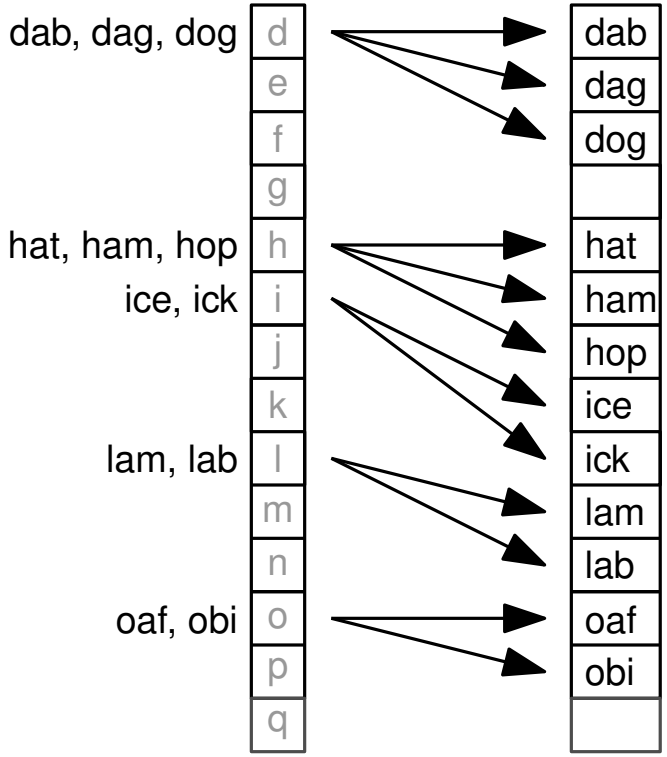
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values

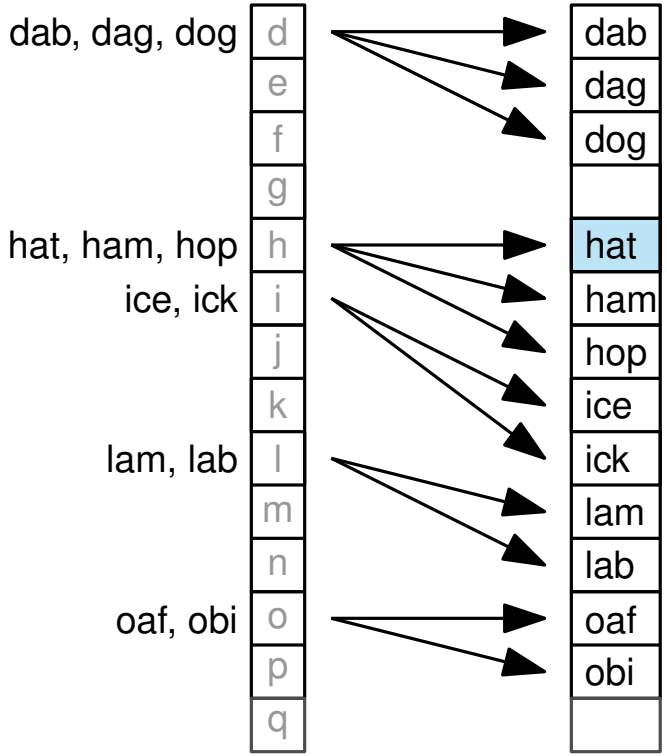


Suche x:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



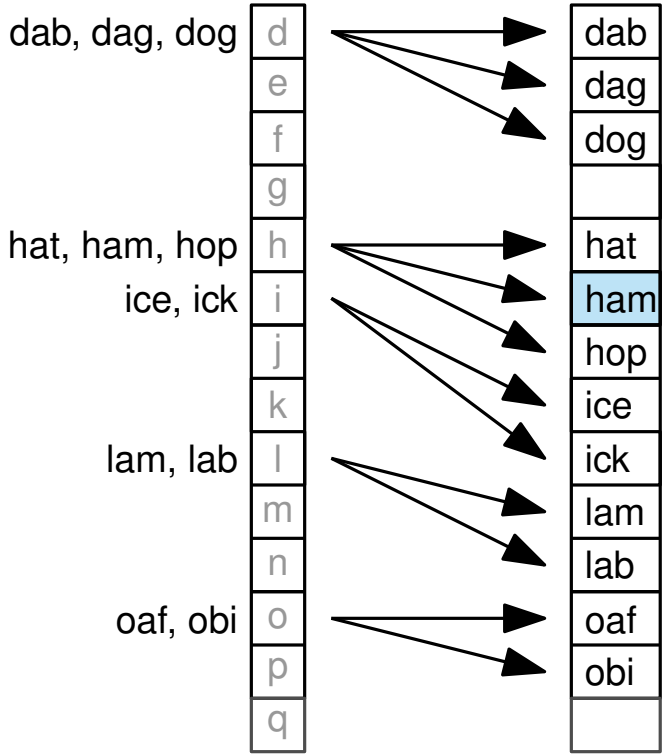
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Beispiel: Suche *hop*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



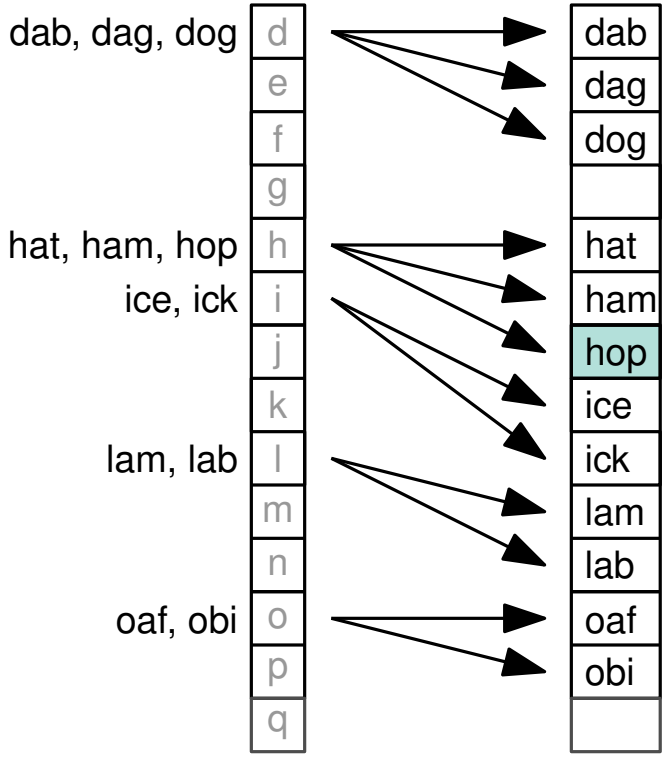
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Beispiel: Suche *hop*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



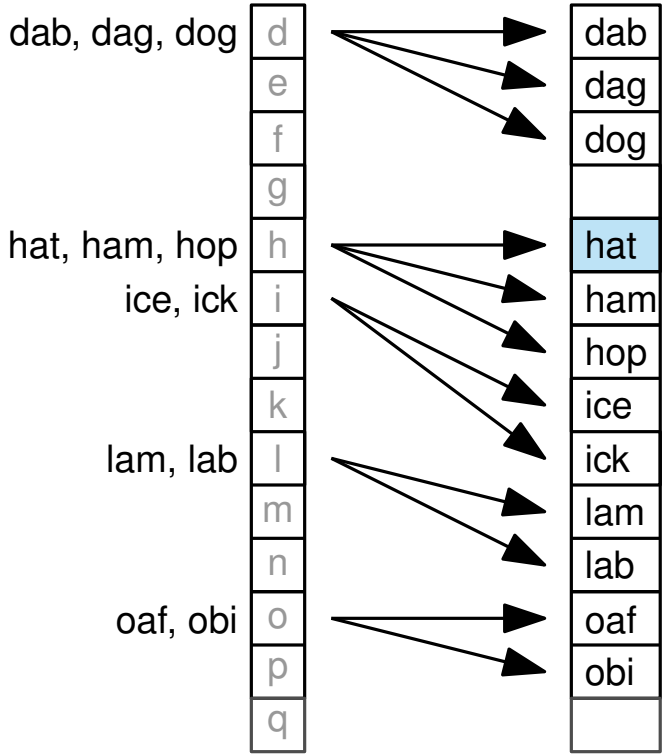
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Beispiel: Suche *hop*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



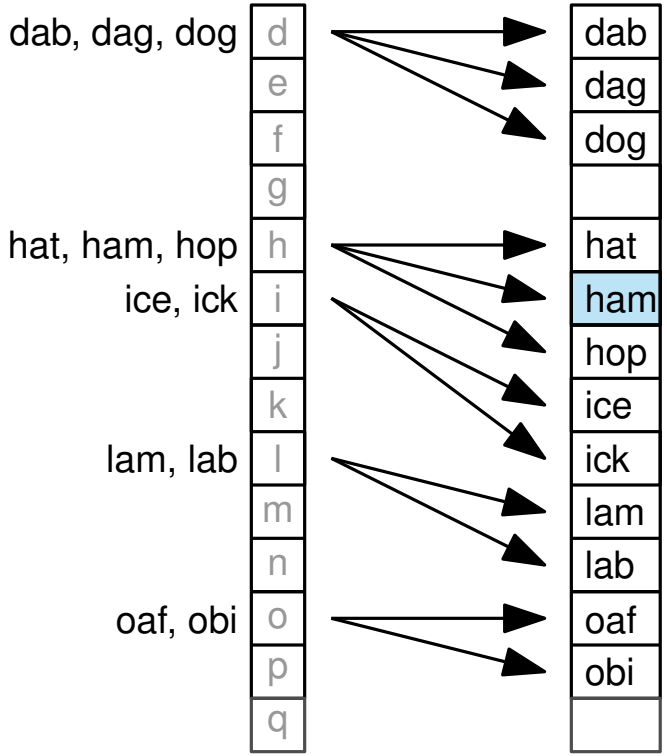
Suche x:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Beispiel: Suche *hit*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



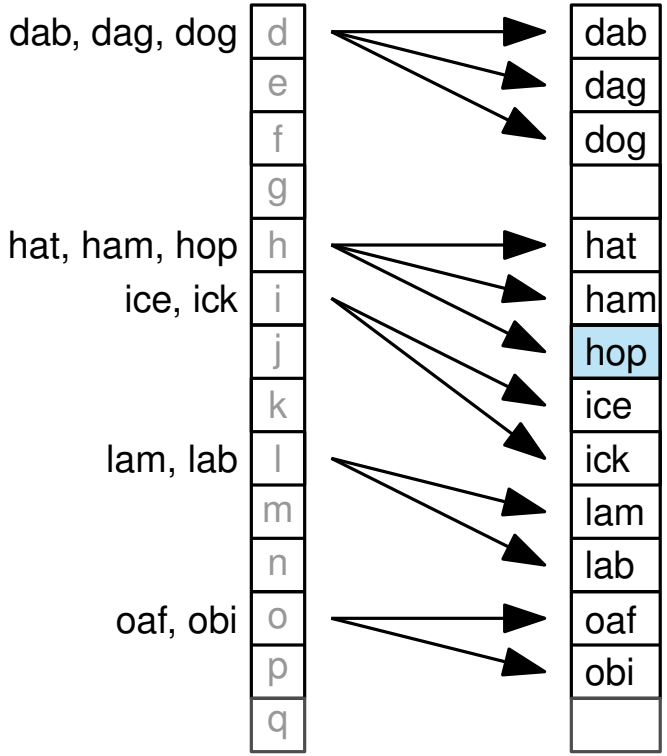
Suche x:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Beispiel: Suche *hit*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



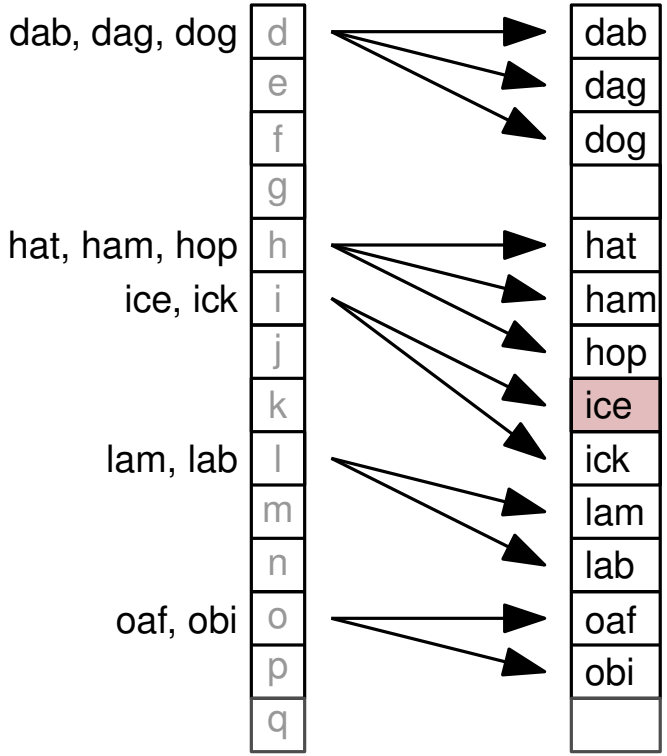
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Beispiel: Suche *hit*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



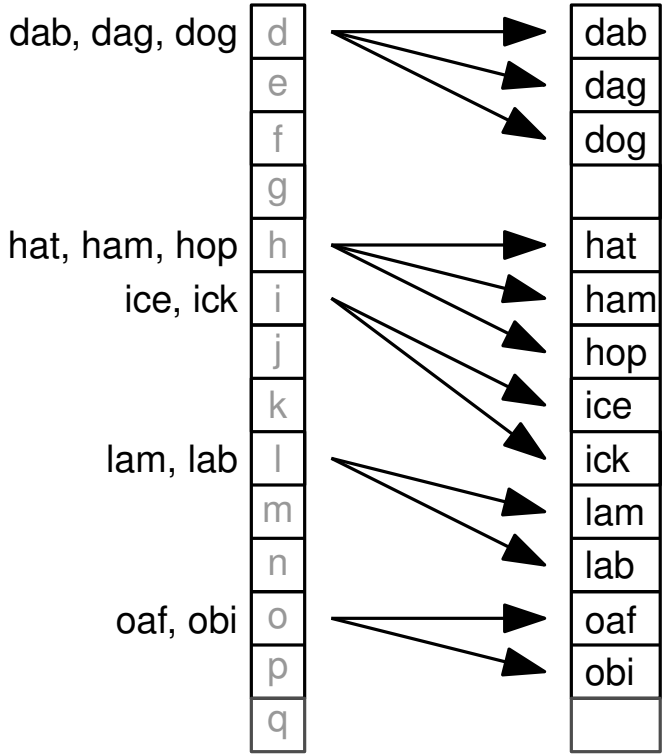
Suche x:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Beispiel: Suche *hit*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



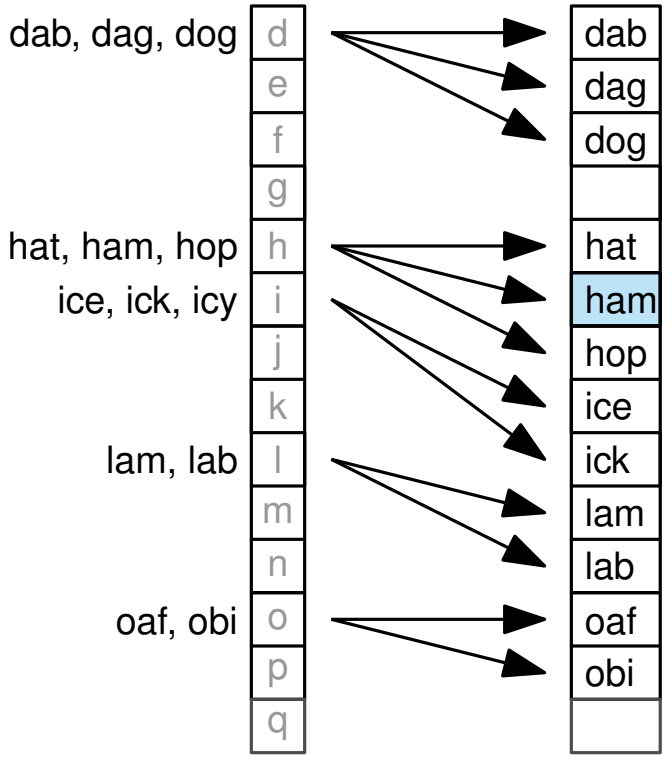
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

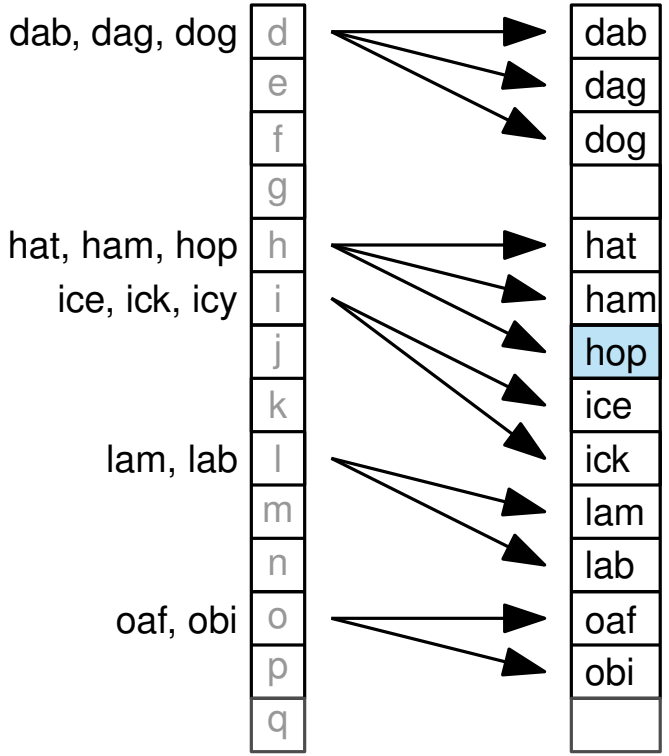
Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Beispiel: Füge *icy* ein

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

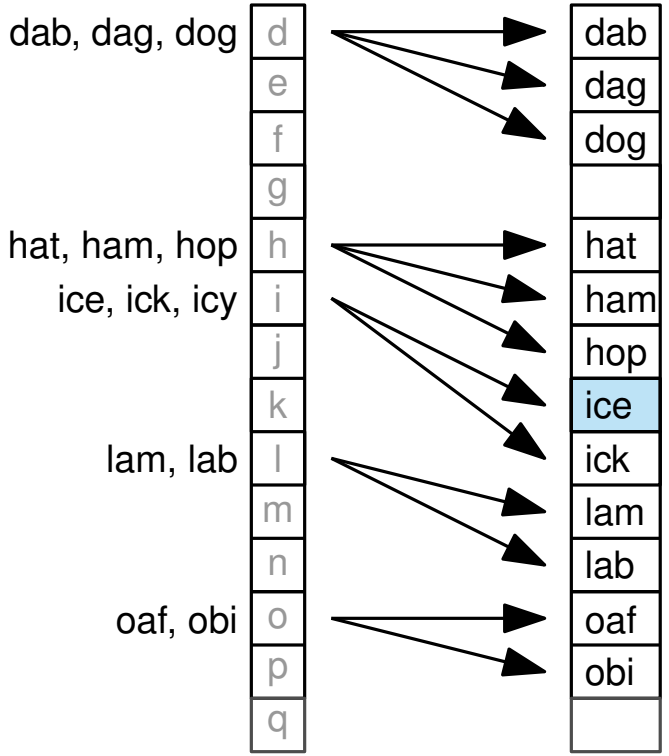
Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Beispiel: Füge *icy* ein

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

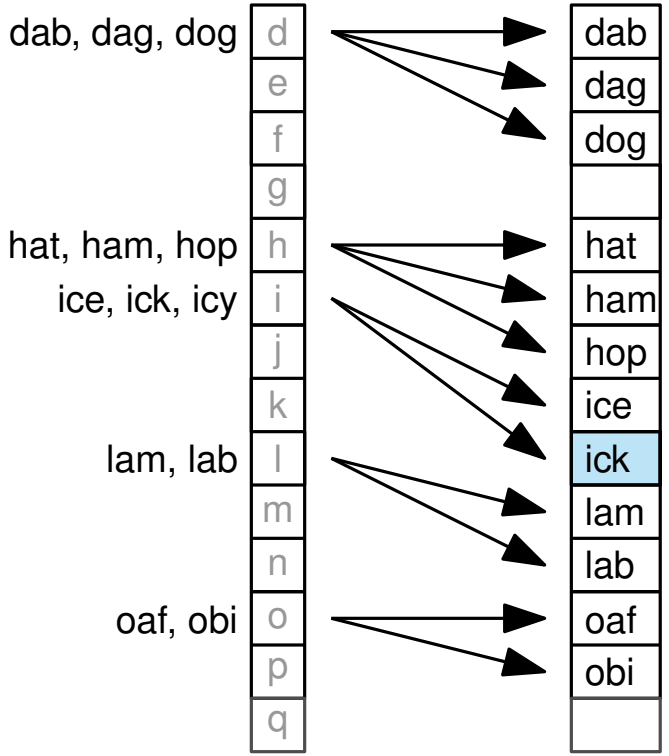
Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Beispiel: Füge *icy* ein

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

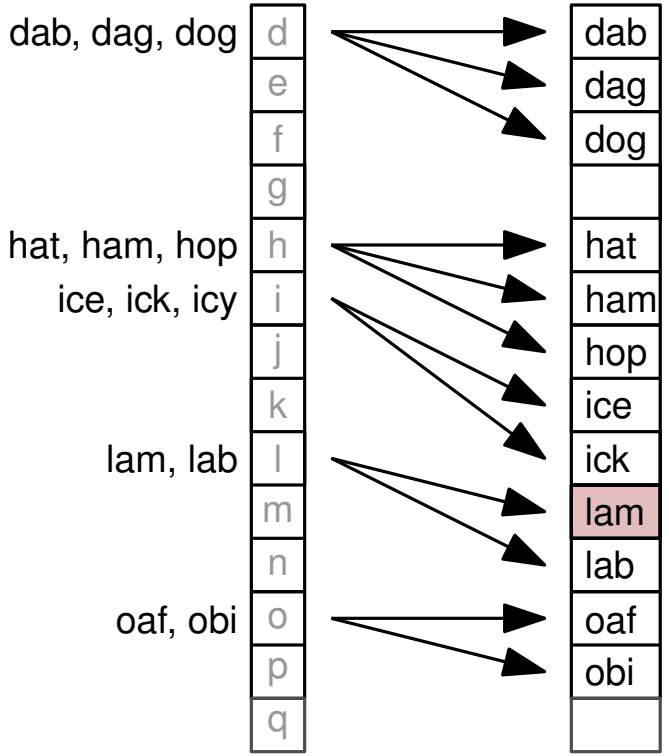
Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Beispiel: Füge *icy* ein

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

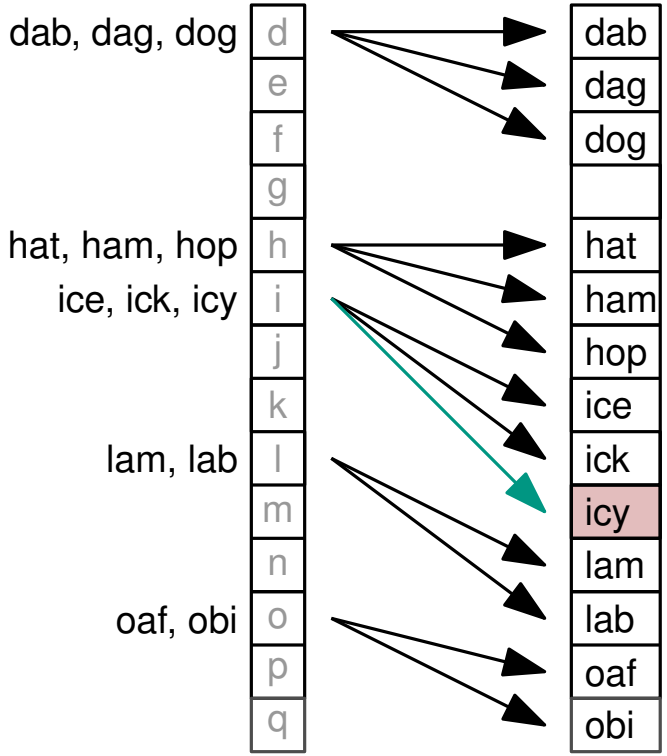
Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Beispiel: Füge *icy* ein

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

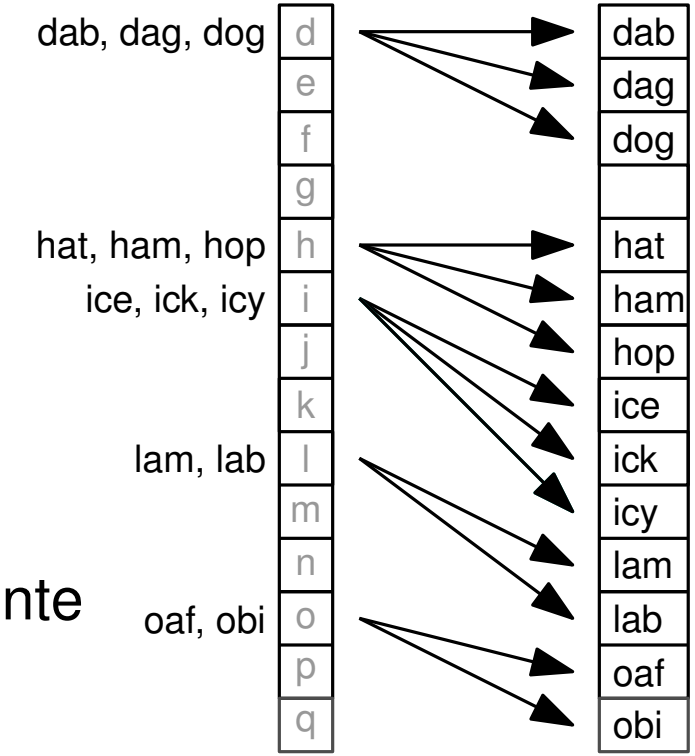
Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Beispiel: Füge *icy* ein

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Lösche *x*:
- Such x und lösche es
- Lass folgenden Elemente nachrücken

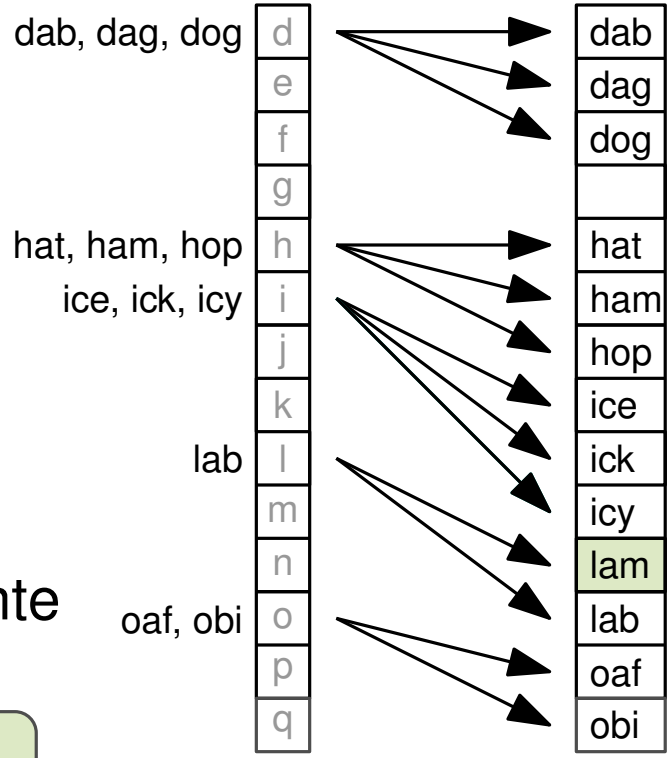
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

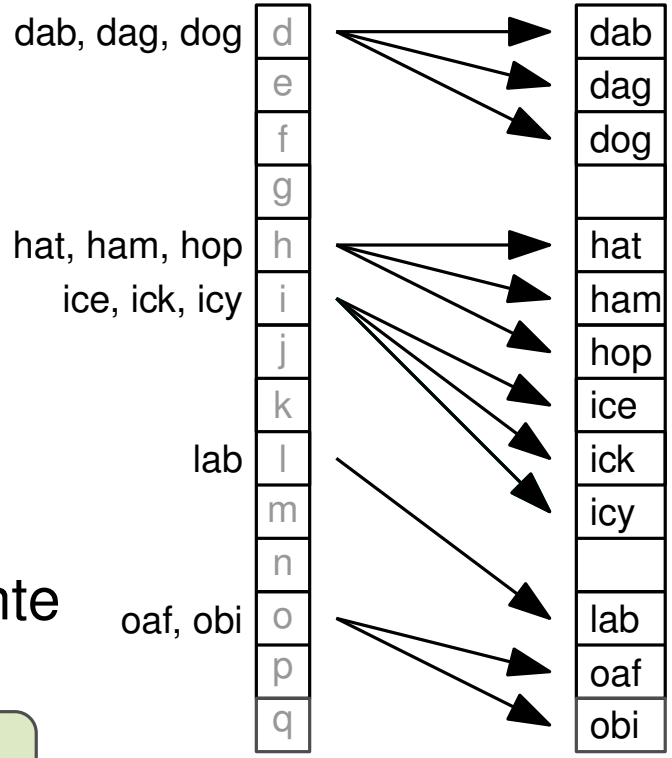
Lösche *x*:
- Such x und lösche es
- Lass folgenden Elemente nachrücken

Beispiel: Lösche *lam*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

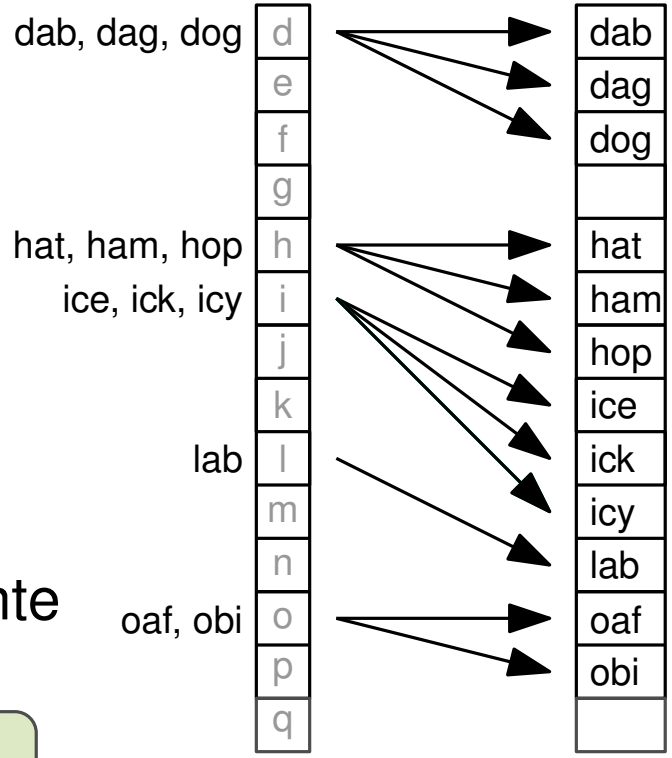
Lösche *x*:
- Such x und lösche es
- Lass folgenden Elemente nachrücken

Beispiel: Lösche *lam*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

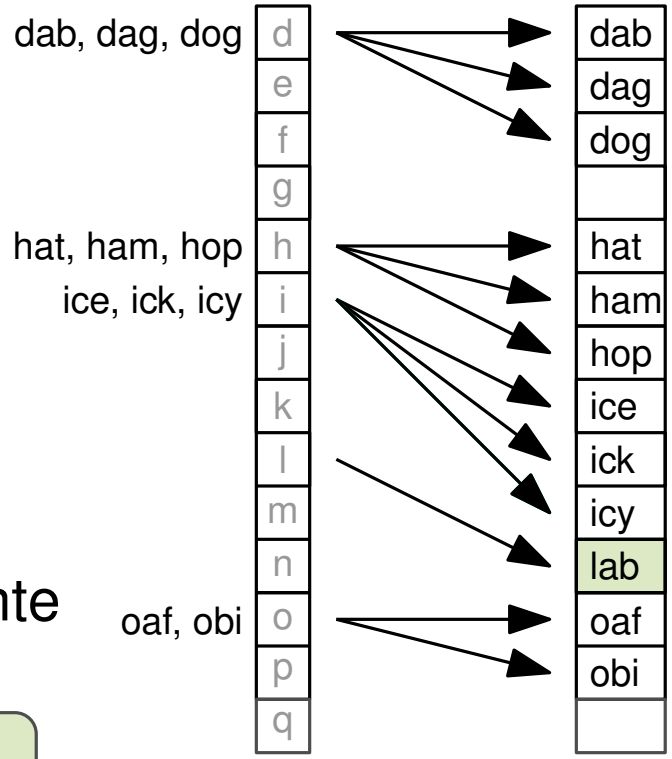
Lösche *x*:
- Such x und lösche es
- Lass folgenden Elemente nachrücken

Beispiel: Lösche *lam*

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Lösche *x*:
- Such x und lösche es
- Lass folgenden Elemente nachrücken

Beispiel: Lösche *lab*

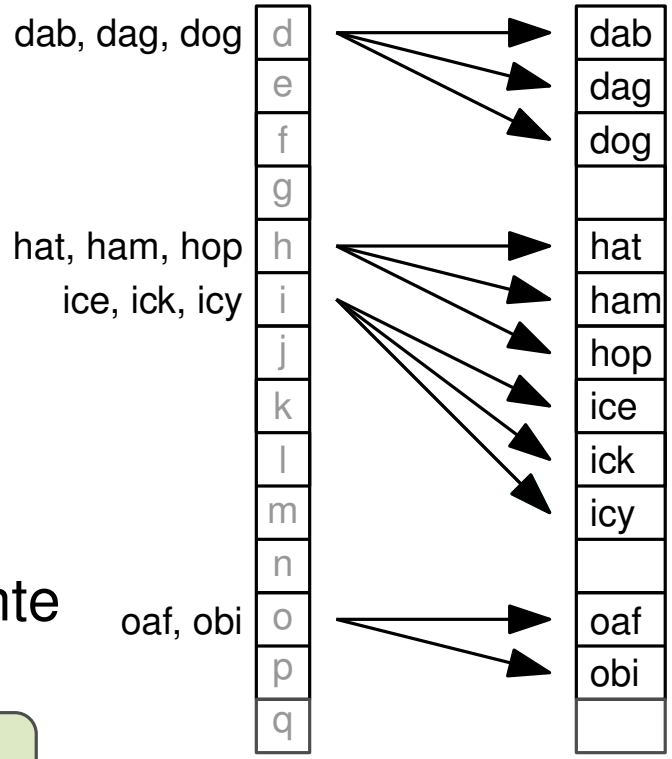
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



Lösche *x*:
- Such x und lösche es
- Lass folgenden Elemente nachrücken

Beispiel: Lösche *lab*

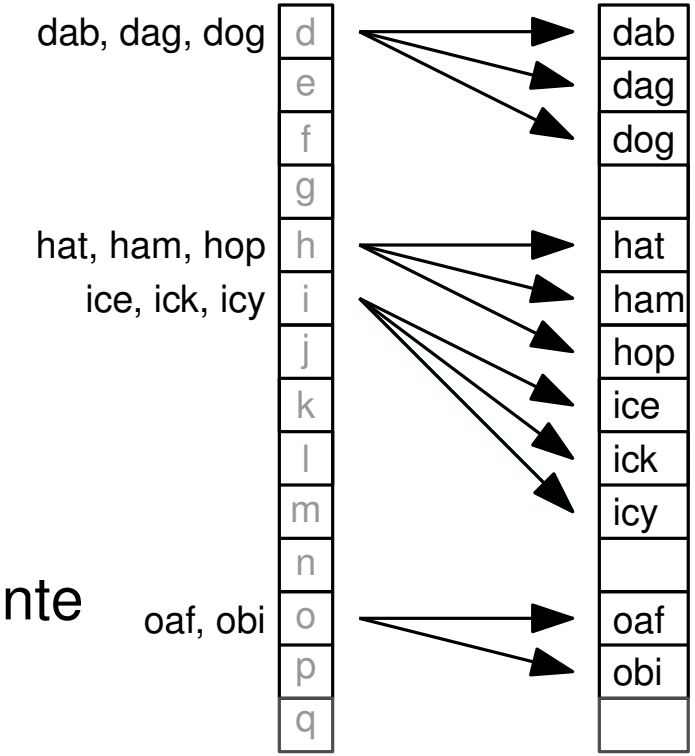
Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

Open Addressing

- Anstatt Werte in Listen zu speichern, wollen wir sie direkt in der Hashtabelle ablegen
- Bei Kollisionen, können wir einen Wert nicht immer an der richtigen Stelle speichern (*open addressing*)

In diesem Beispiel:
- Nutze ersten Buchstaben als Hashfunktion
- keys = values



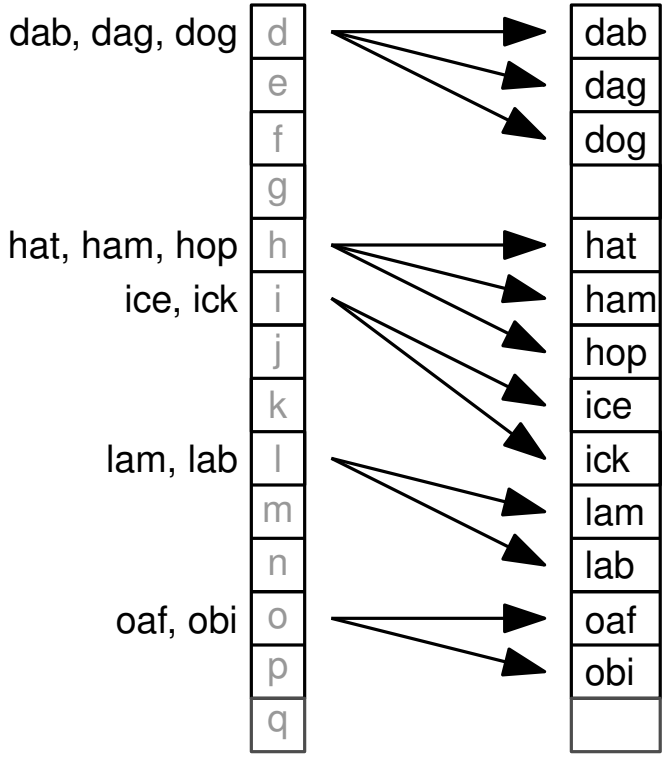
Lösche *x*:
- Such x und lösche es
- Lass folgenden Elemente nachrücken

Suche *x*:
- Starte bei Index $i = h(x)$ und inkrementiere i
- Breche ab, sobald man x gefunden hat oder hinter seinem Bucket ist

Füge *x* ein:
- Füge x am Ende seines Buckets ein
- Verschiebe alle nachfolgenden Elemente nach hinten

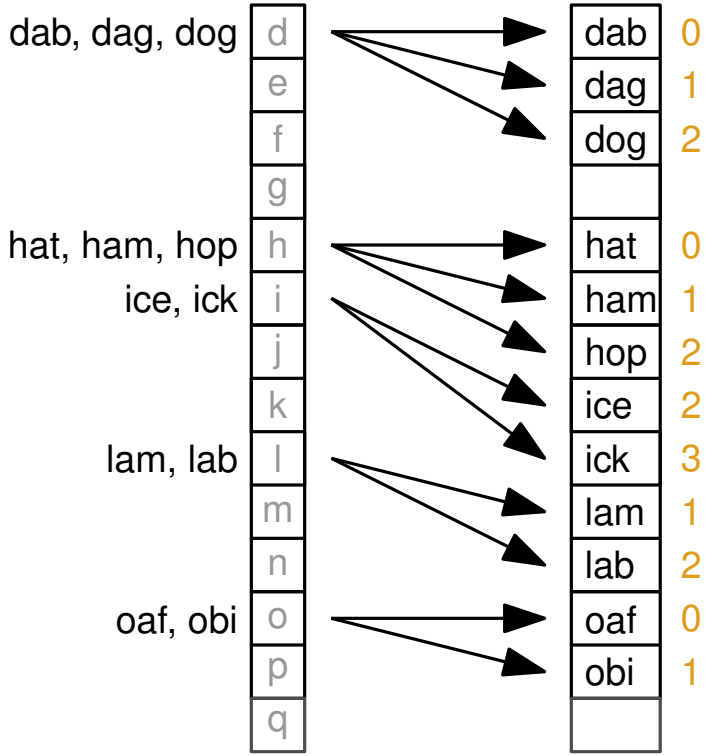
Robin Hood Hashing

- Wie implementieren wir das Verfahren?



Robin Hood Hashing

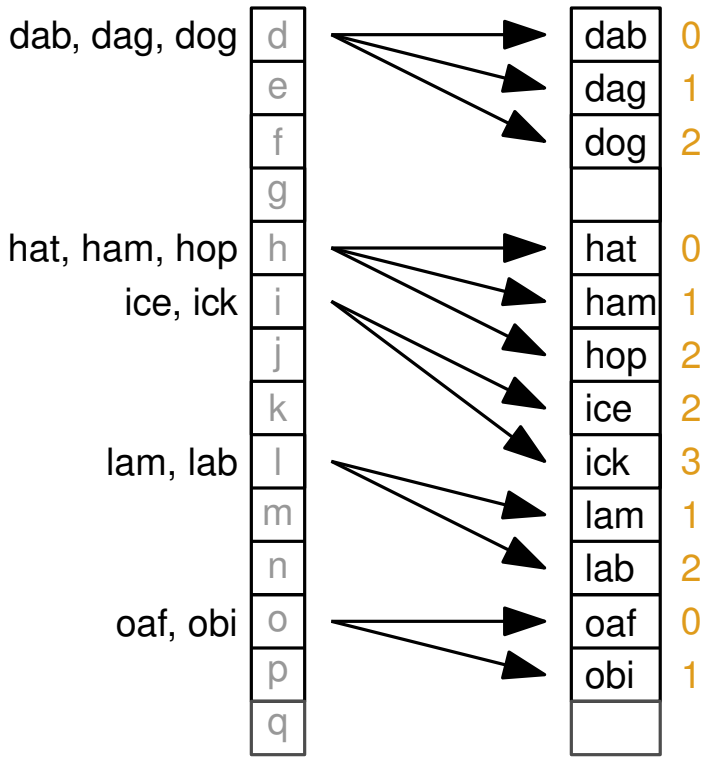
- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist



Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

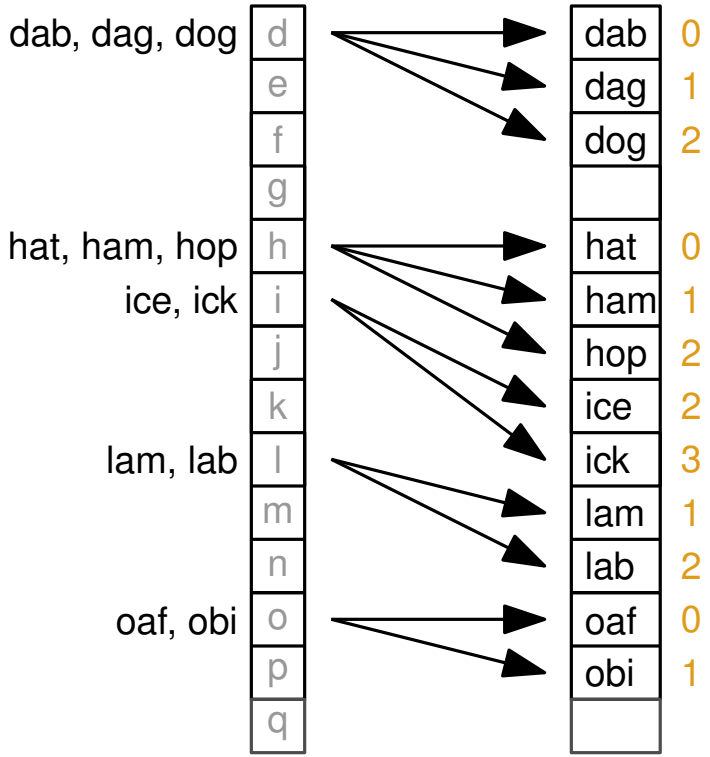
- Weit weg von zuhause = arm
- Nahe zuhause = reich



Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



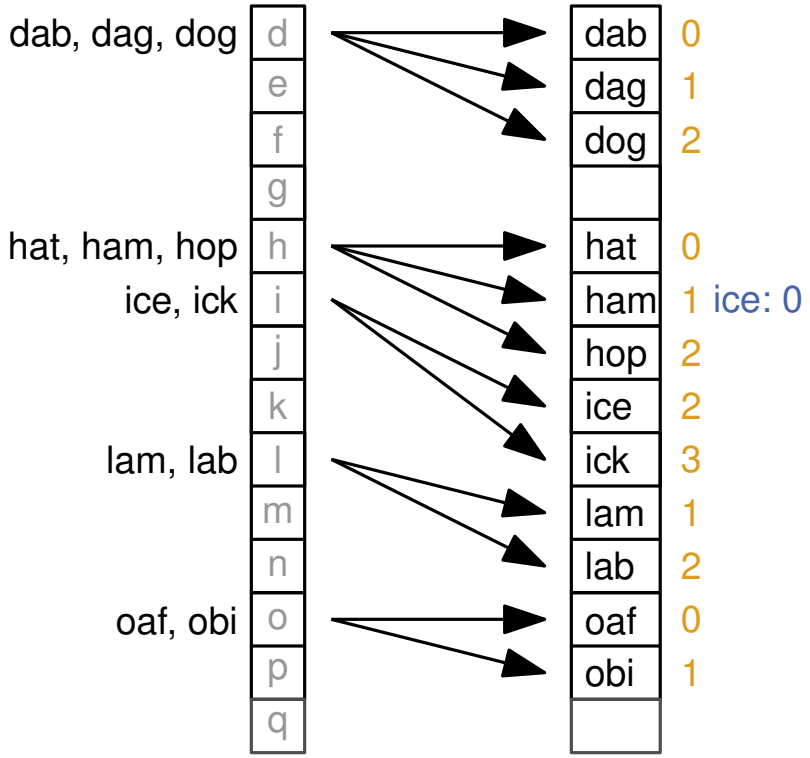
Suche x:

- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche *x*:

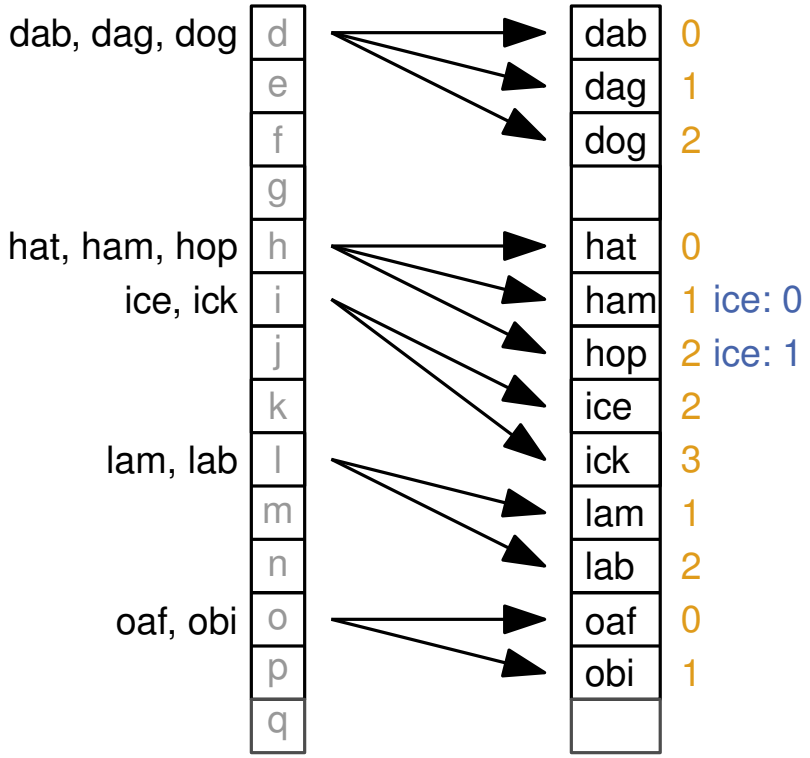
- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Beispiel: suche *ice*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche *x*:

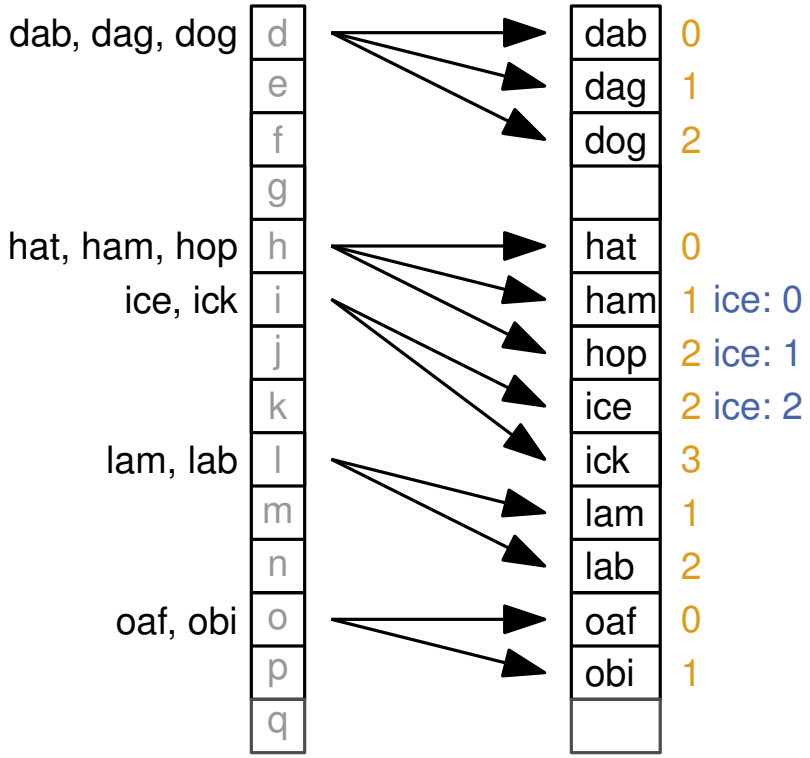
- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Beispiel: suche *ice*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche x:

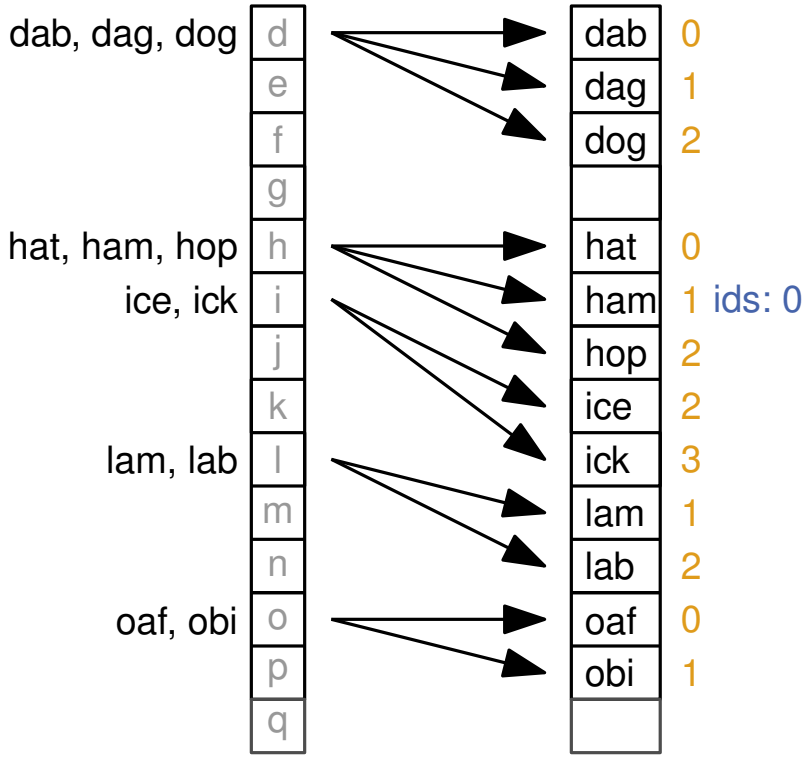
- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Beispiel: suche *ice*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche *x*:

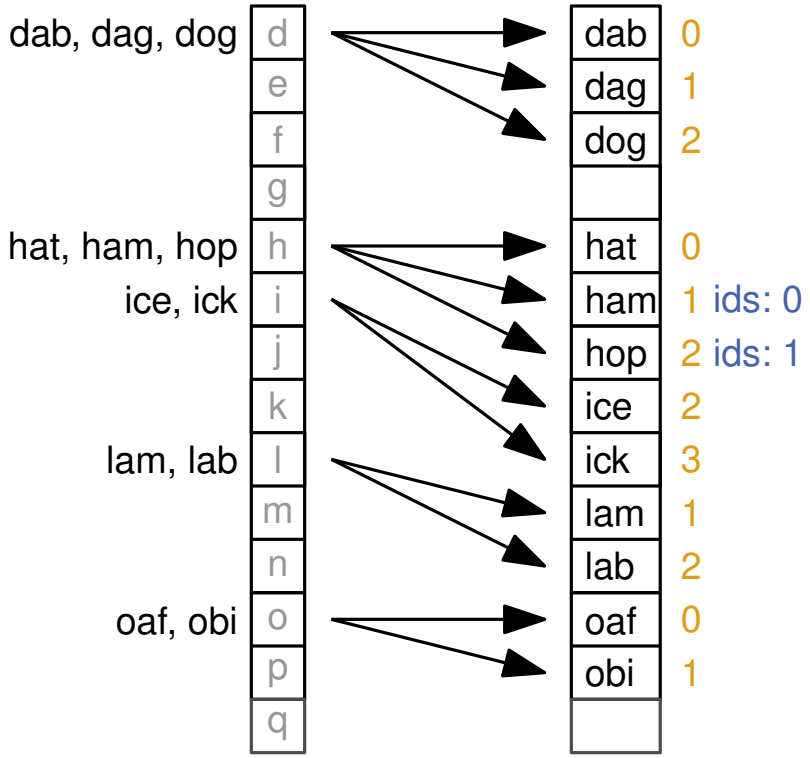
- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Beispiel: suche *ids*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche x:

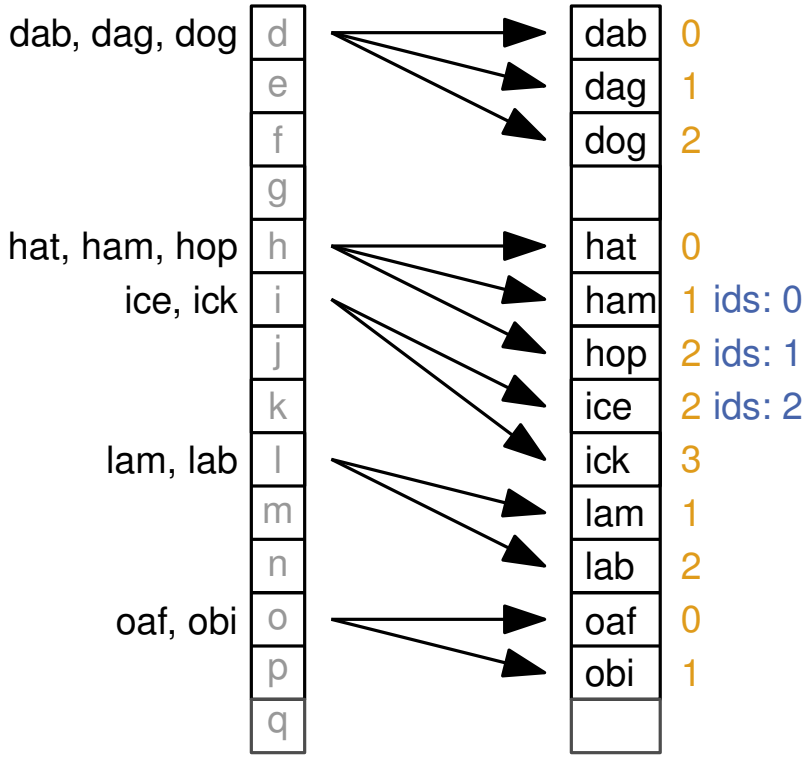
- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Beispiel: suche *ids*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche x:

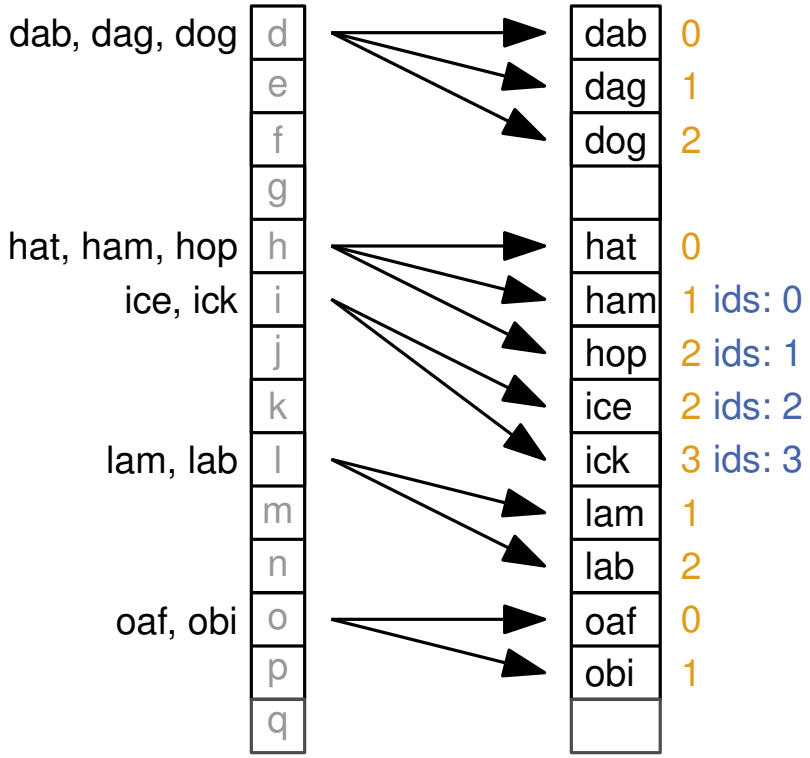
- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Beispiel: suche *ids*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche x:

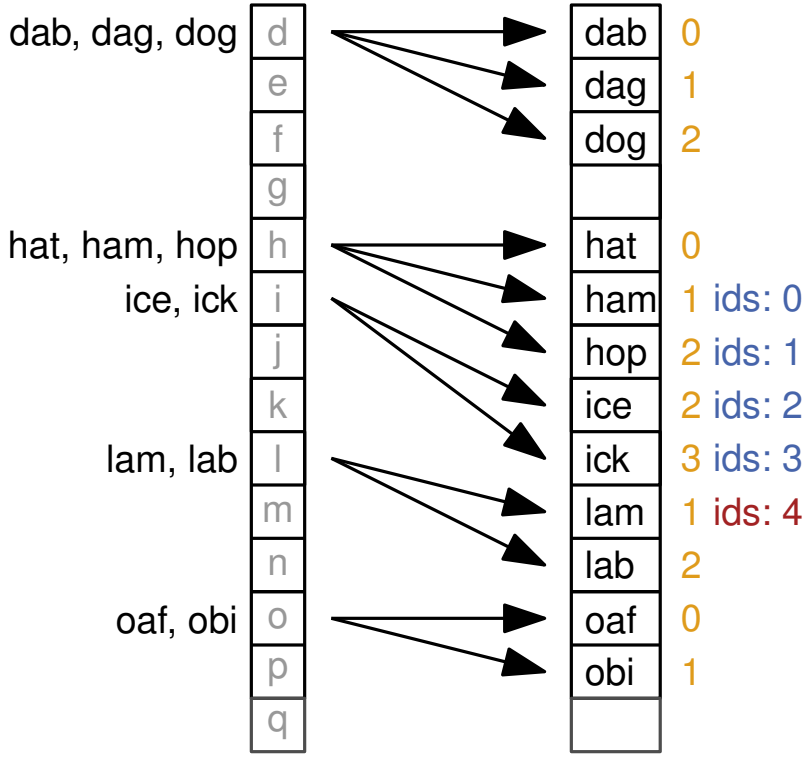
- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Beispiel: suche *ids*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich



Suche x:

- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

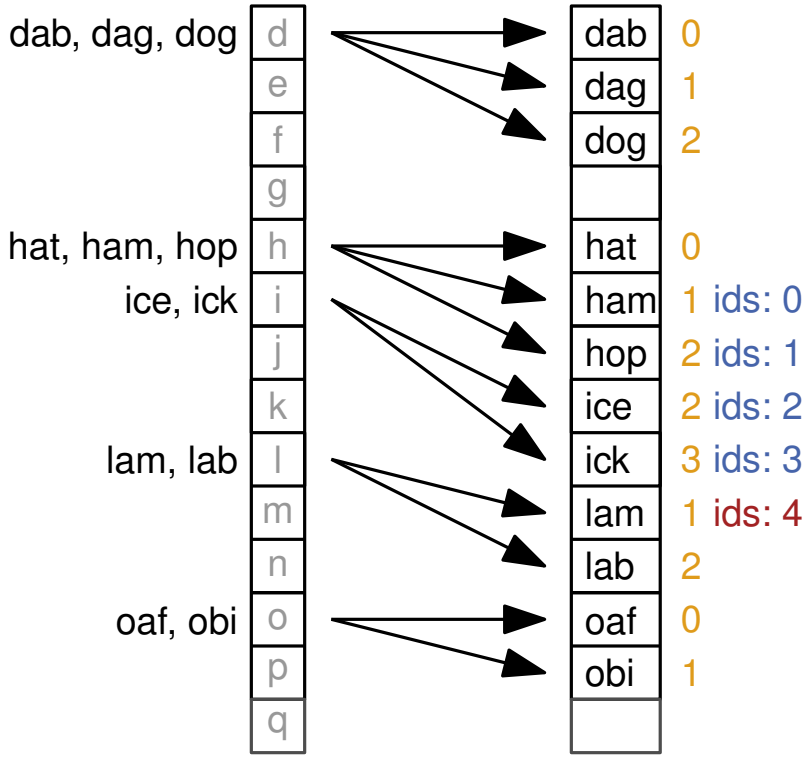
Beispiel: suche *ids*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket



Suche x:

- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

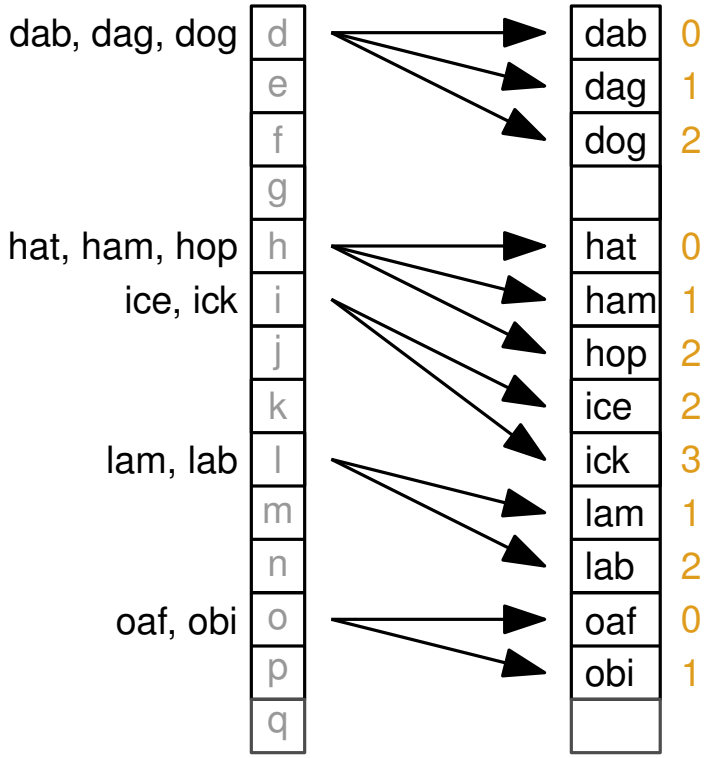
Beispiel: suche *ids*

Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei i reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

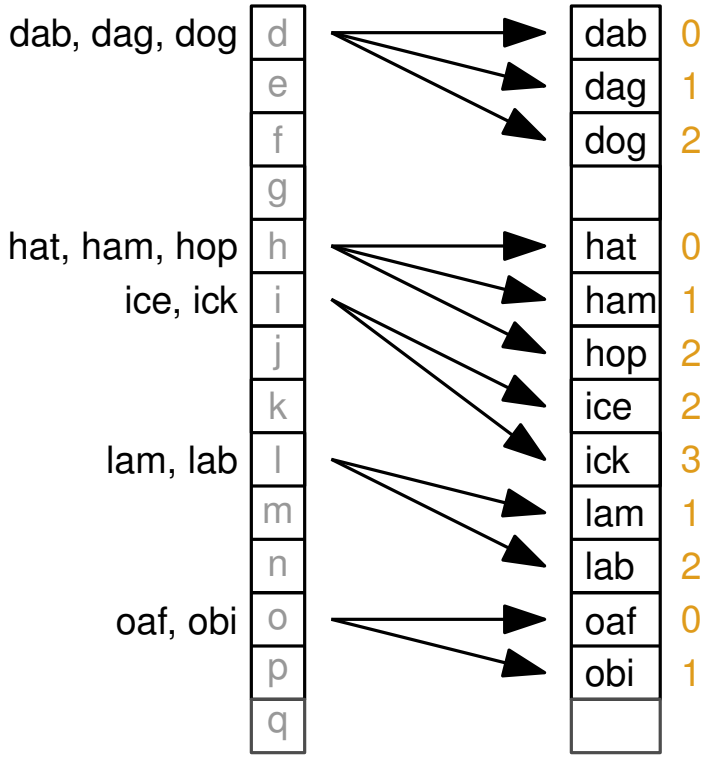
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

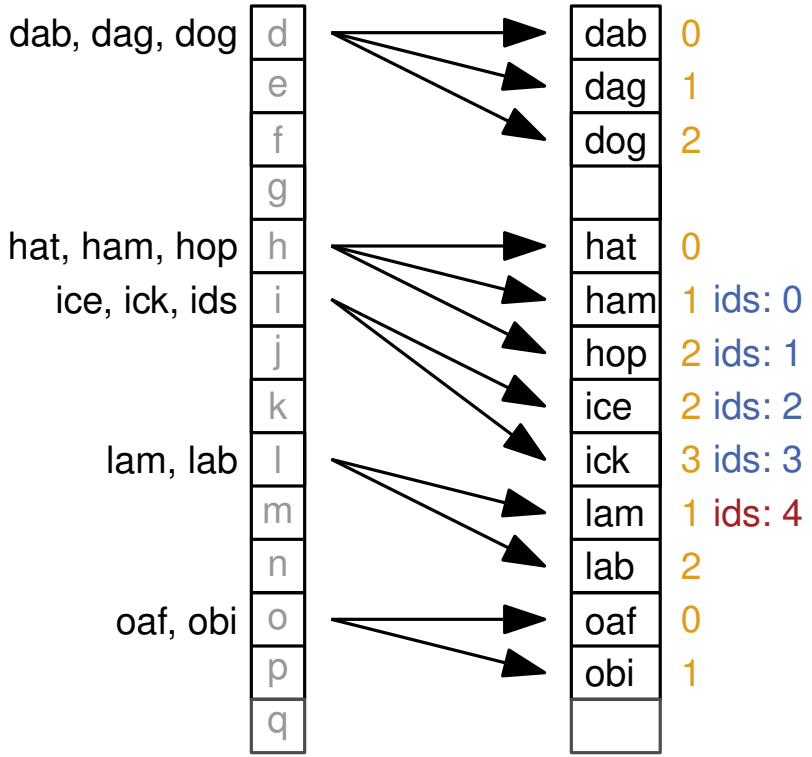
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

Beispiel: Füge *ids* ein

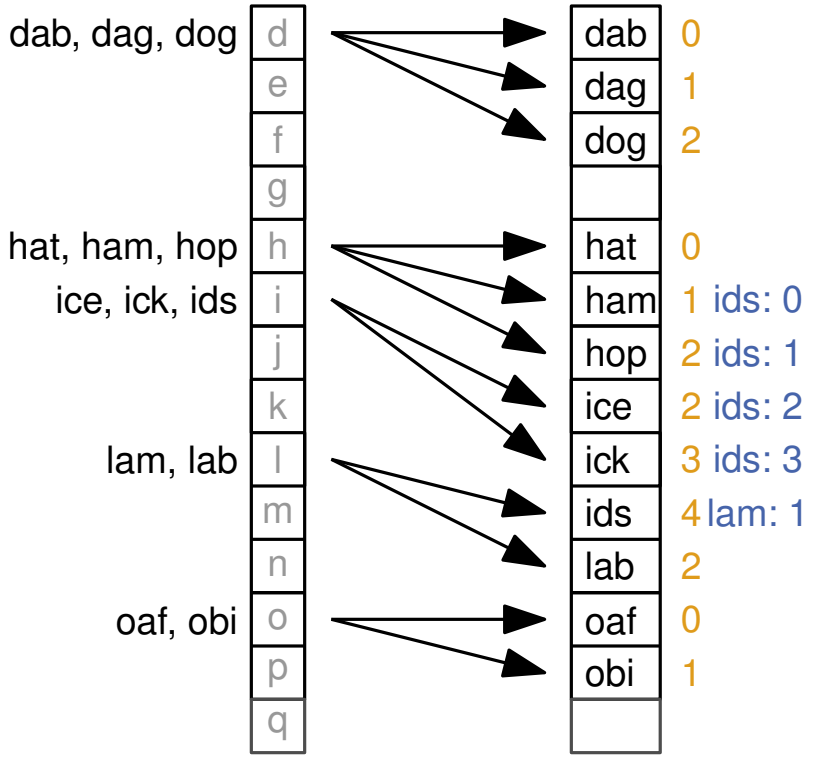
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

Beispiel: Füge *ids* ein

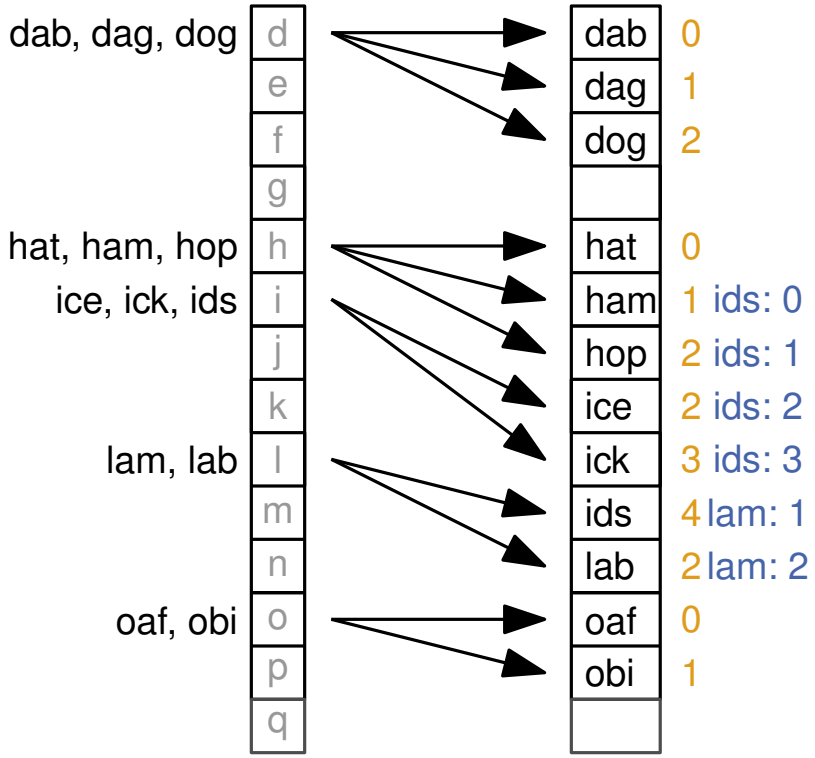
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

Beispiel: Füge *ids* ein

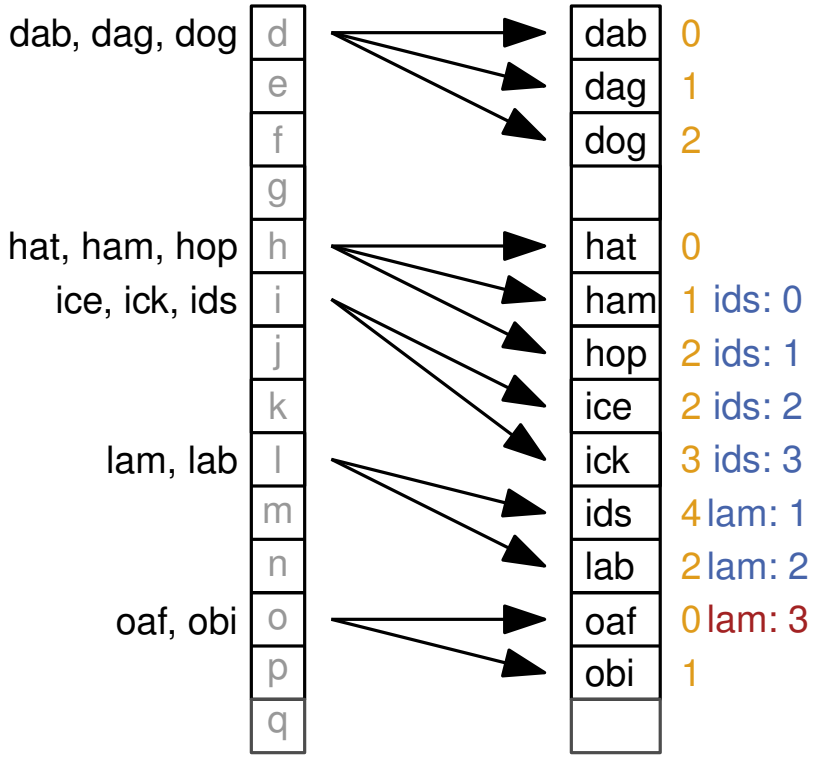
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

Beispiel: Füge *ids* ein

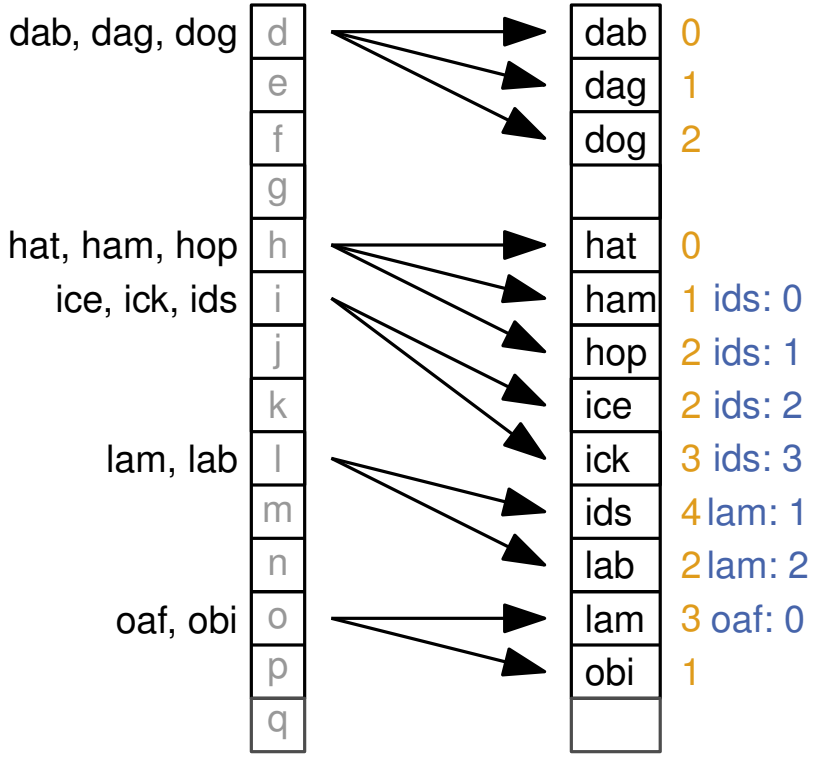
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

Beispiel: Füge *ids* ein

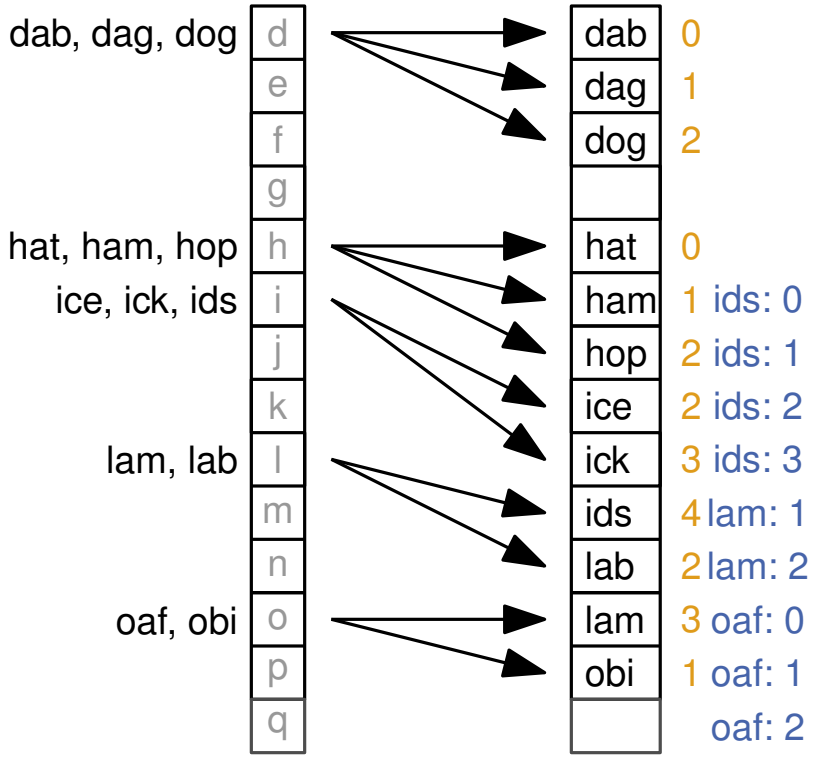
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

Beispiel: Füge *ids* ein

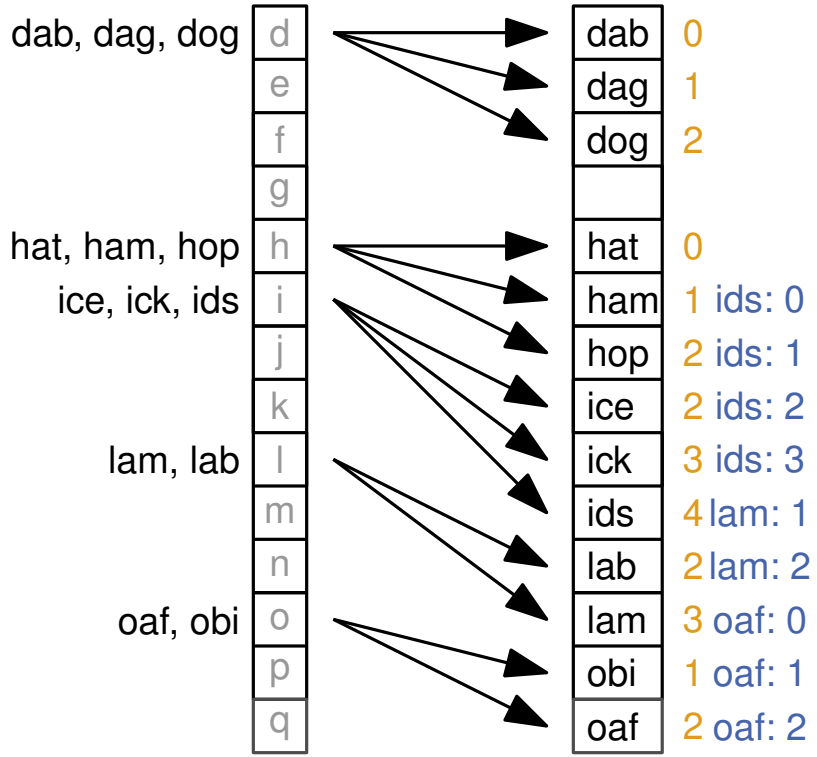
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Suche *x*:

- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

Beispiel: Füge *ids* ein

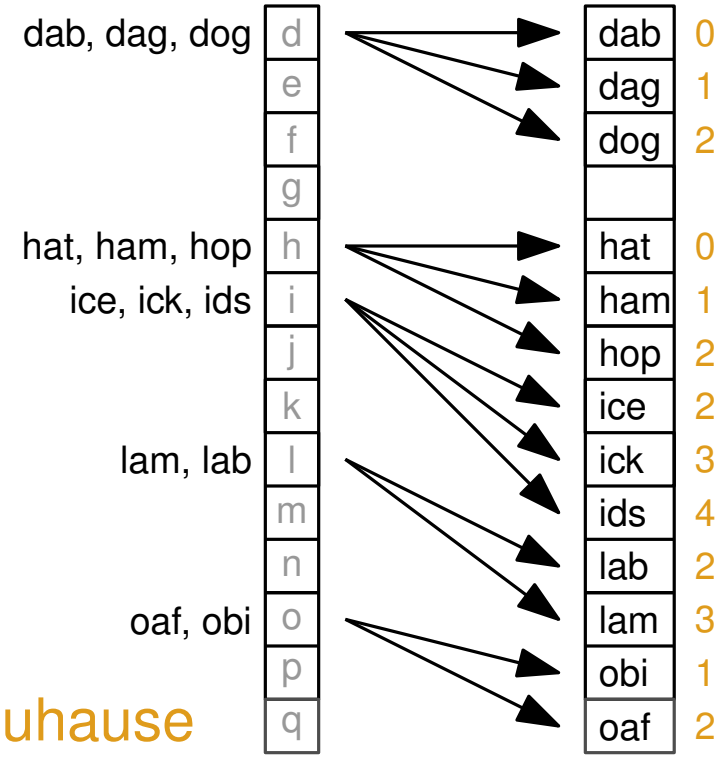
Robin Hood Hashing

- Wie implementieren wir das Verfahren?
- Speichere zusätzlich für jeden Eintrag, wie weit entfernt der Wert von seiner eigentlichen Position weg ist

- Weit weg von zuhause = arm
- Nahe zuhause = reich

- Ärmer als x = vorheriger Bucket
- Reicher als x = nächster Bucket

Robin Hood gibt den Armen und nimmt den Reichen!



Lösche *x*:

- Suche und lösche *x*
- Shifte alles was nicht zuhause ist eins nach oben

Suche *x*:

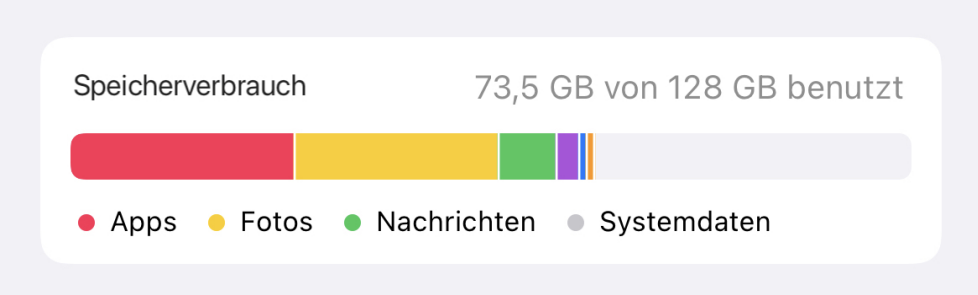
- Starte bei $i = h(x)$
- Wenn Wert bei *i* reicher als x an der stelle wäre, breche ab

Füge *x* ein:

- Finde den ersten reicheren Wert und tausche mit diesem
- Fahre fort, bis eine Lücke gefunden wurde

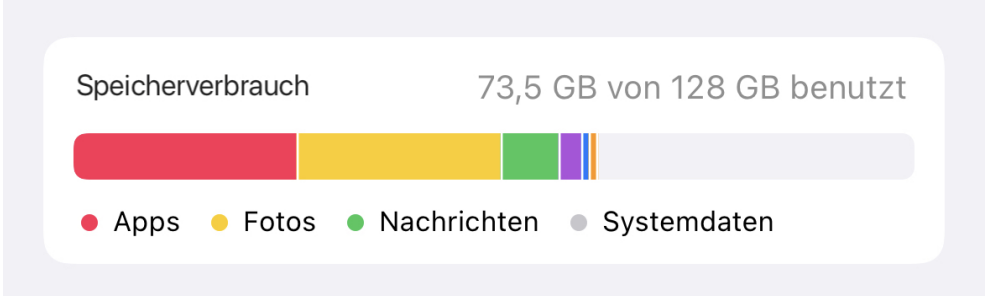
Platz sparen

- Problem: Smartphone-Speicher ist knapp



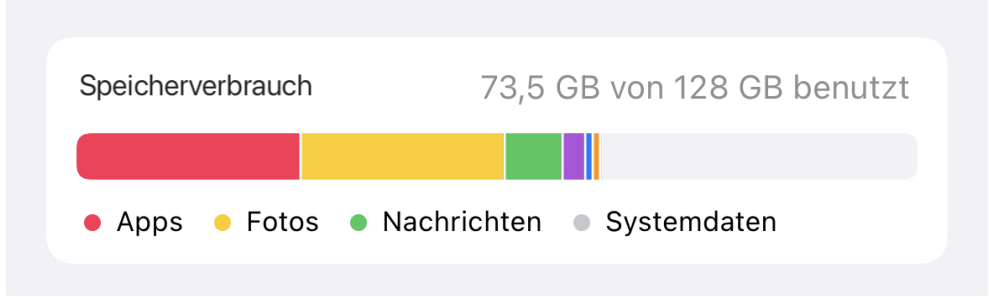
Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein



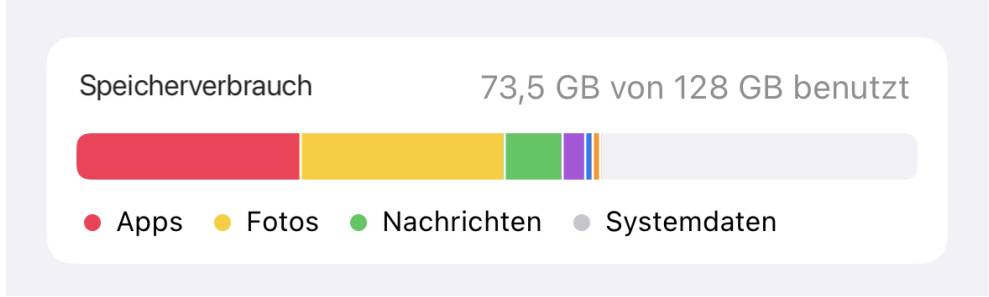
Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten



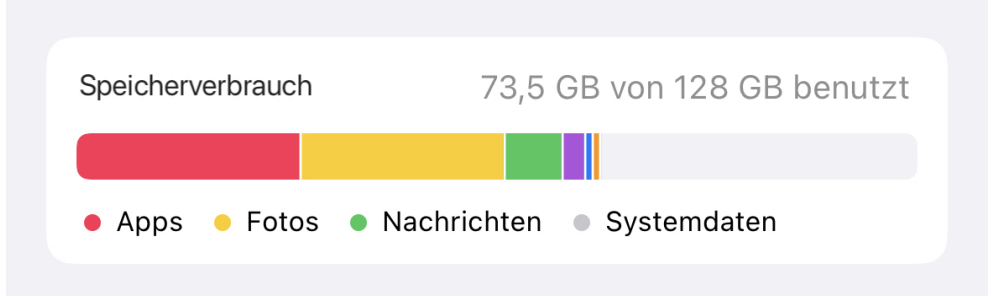
Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?



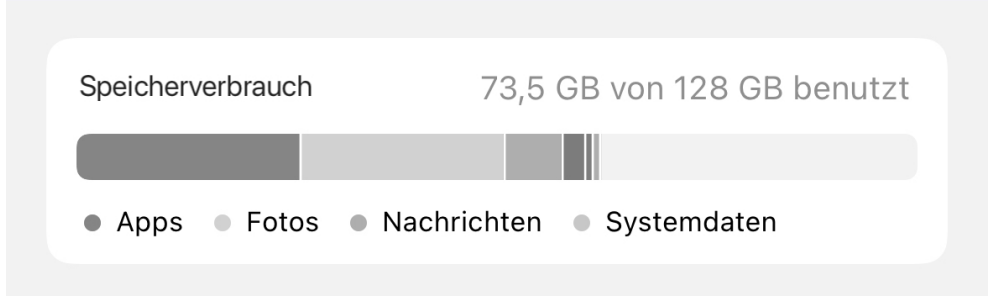
Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.



Platz sparen

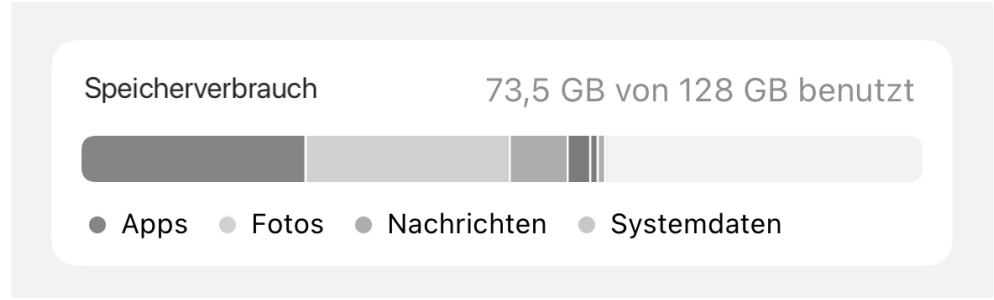
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen



Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$

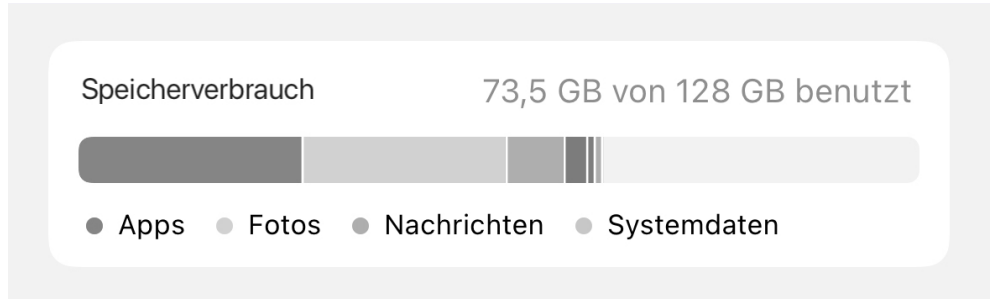
57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57



Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array

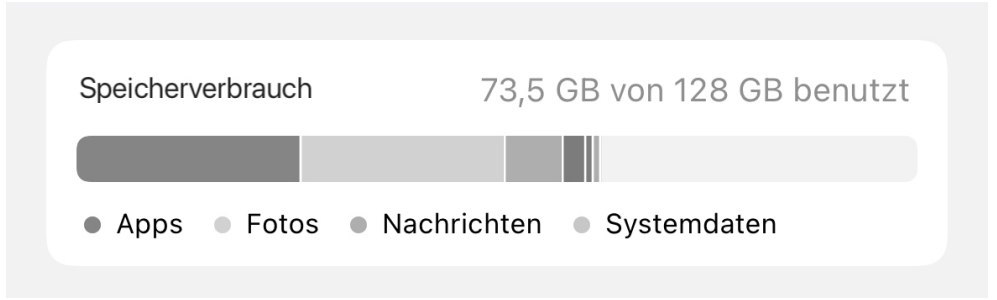
57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57



Platz sparen

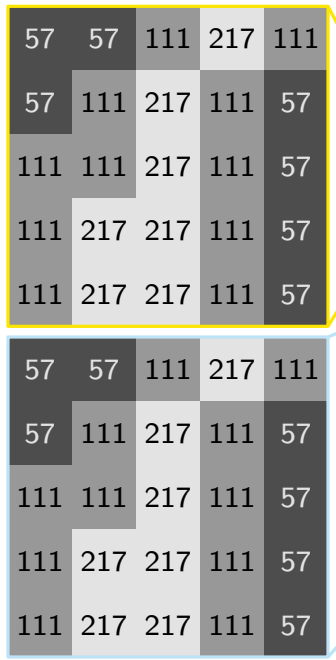
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich

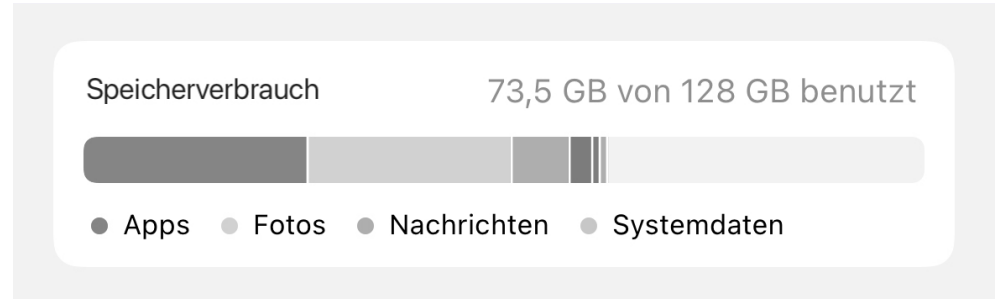
57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57



Platz sparen

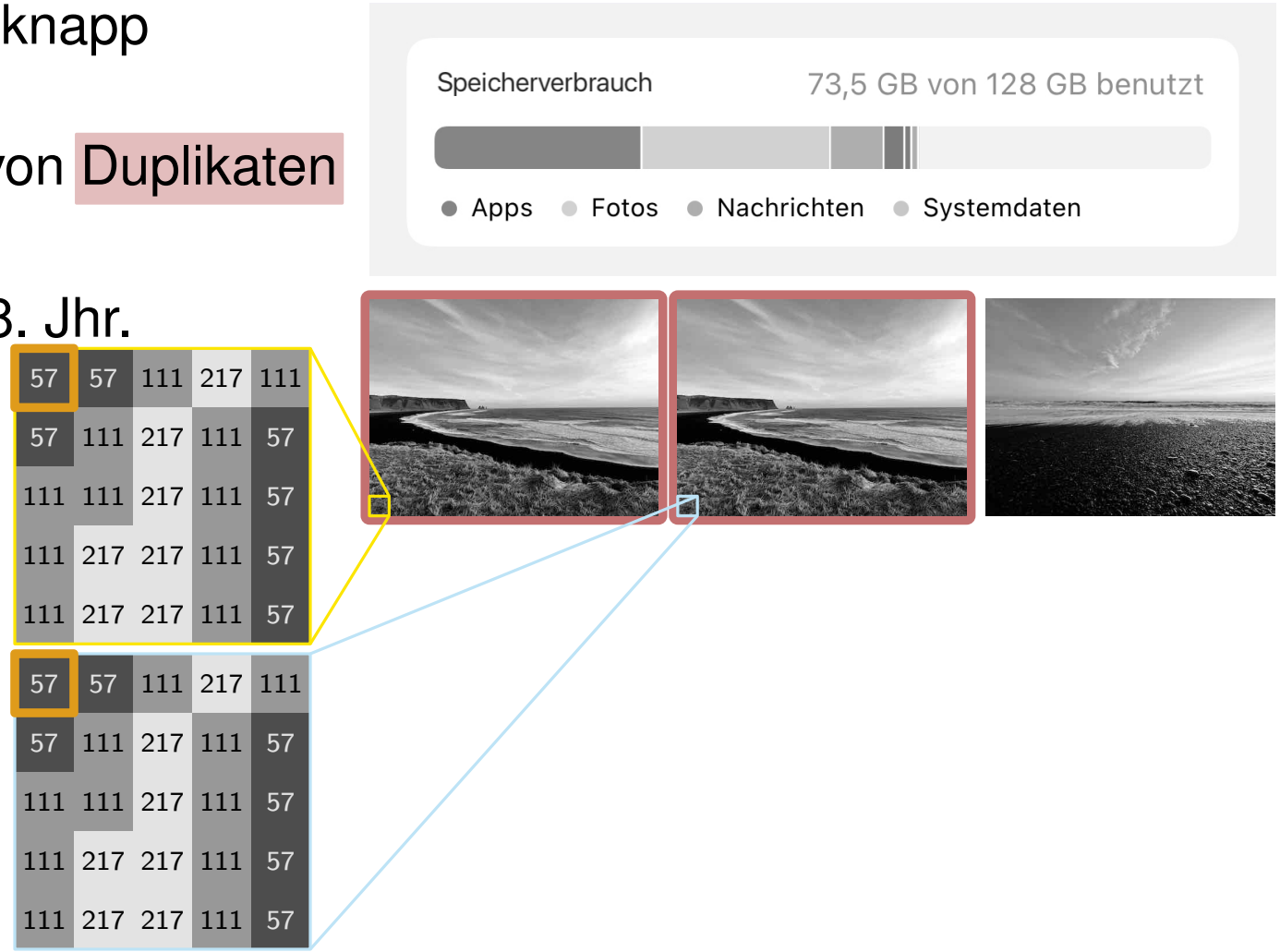
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit





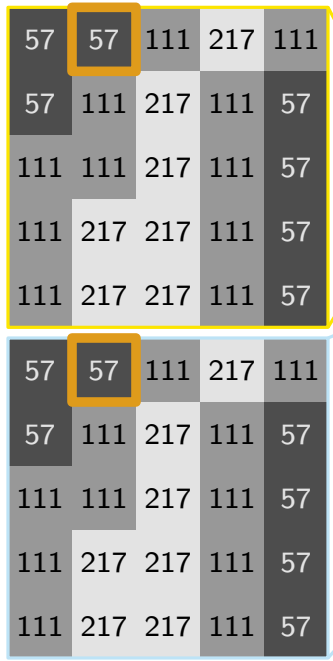
Platz sparen

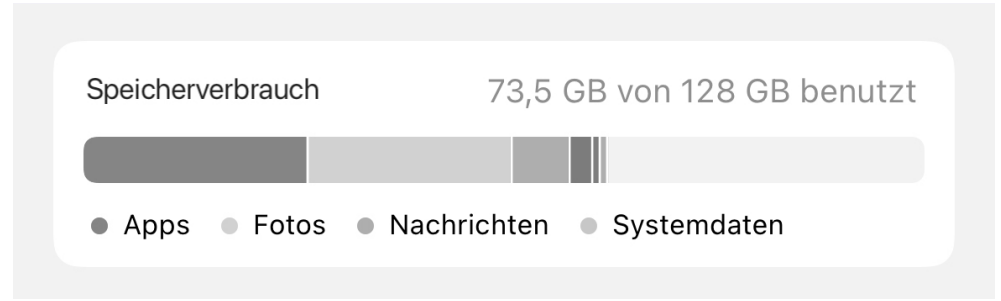
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit



Platz sparen

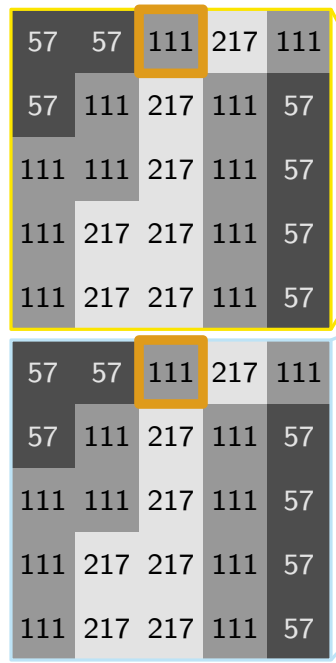
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit

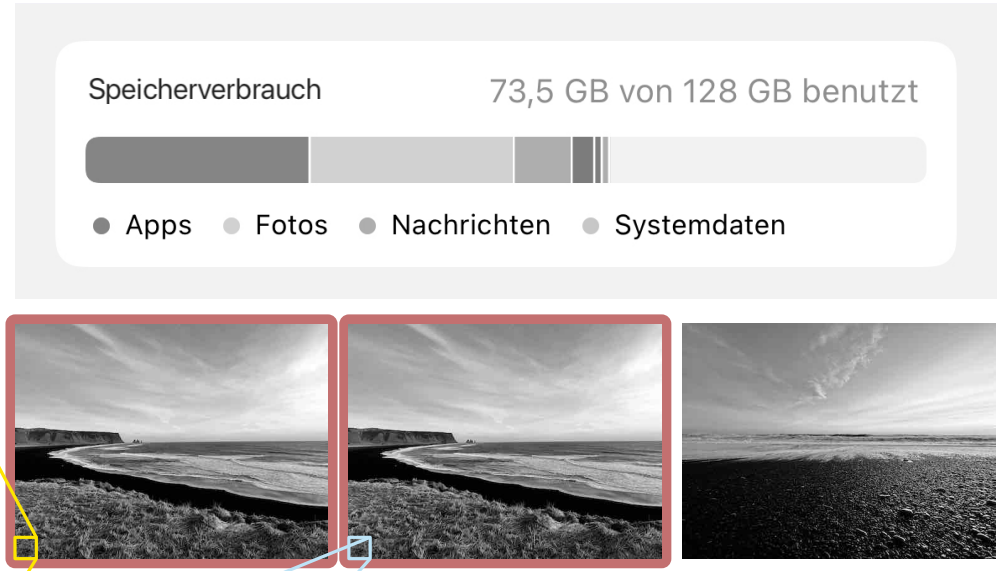




Platz sparen

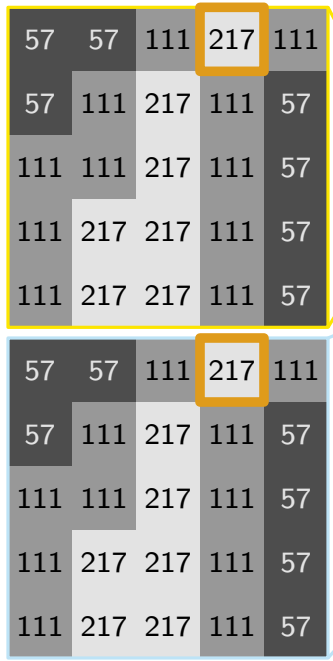
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit

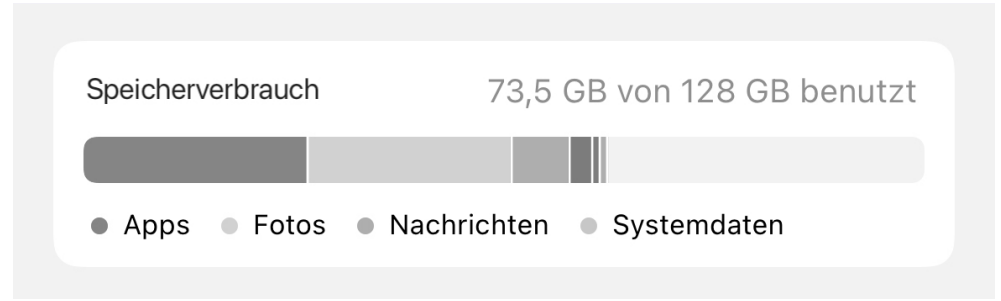




Platz sparen

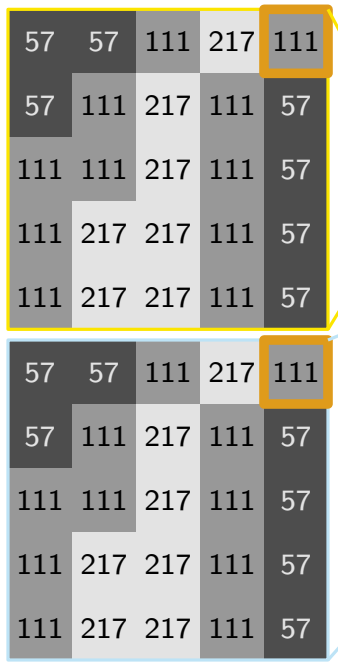
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit

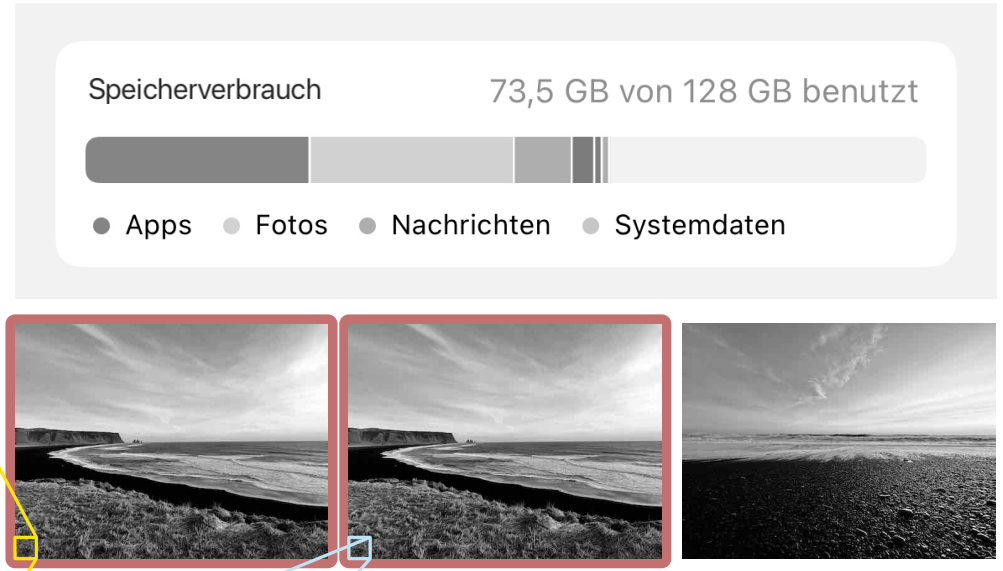




Platz sparen

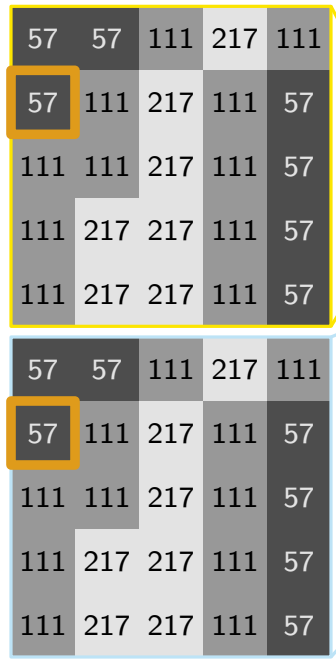
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit

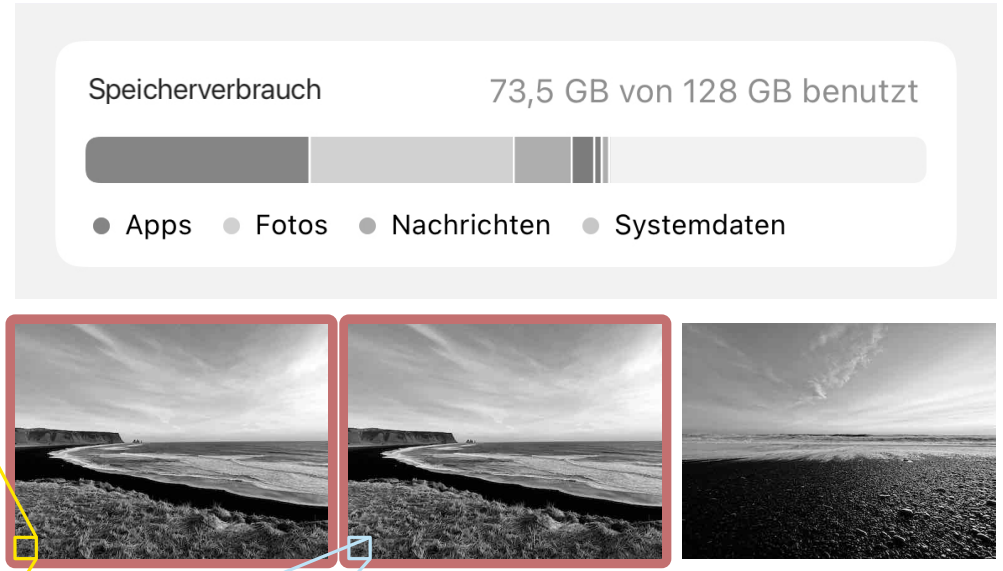




Platz sparen

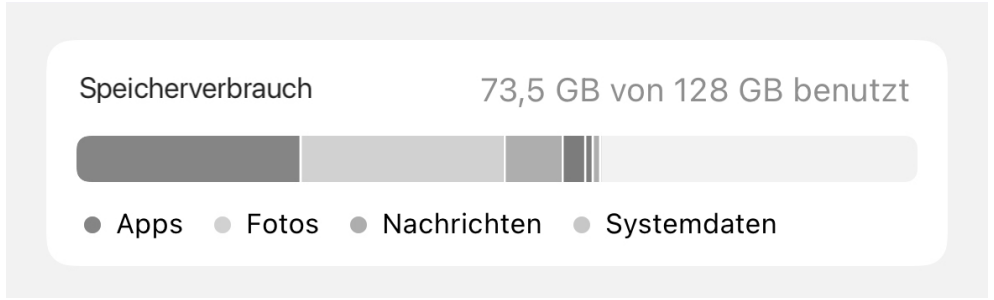
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit

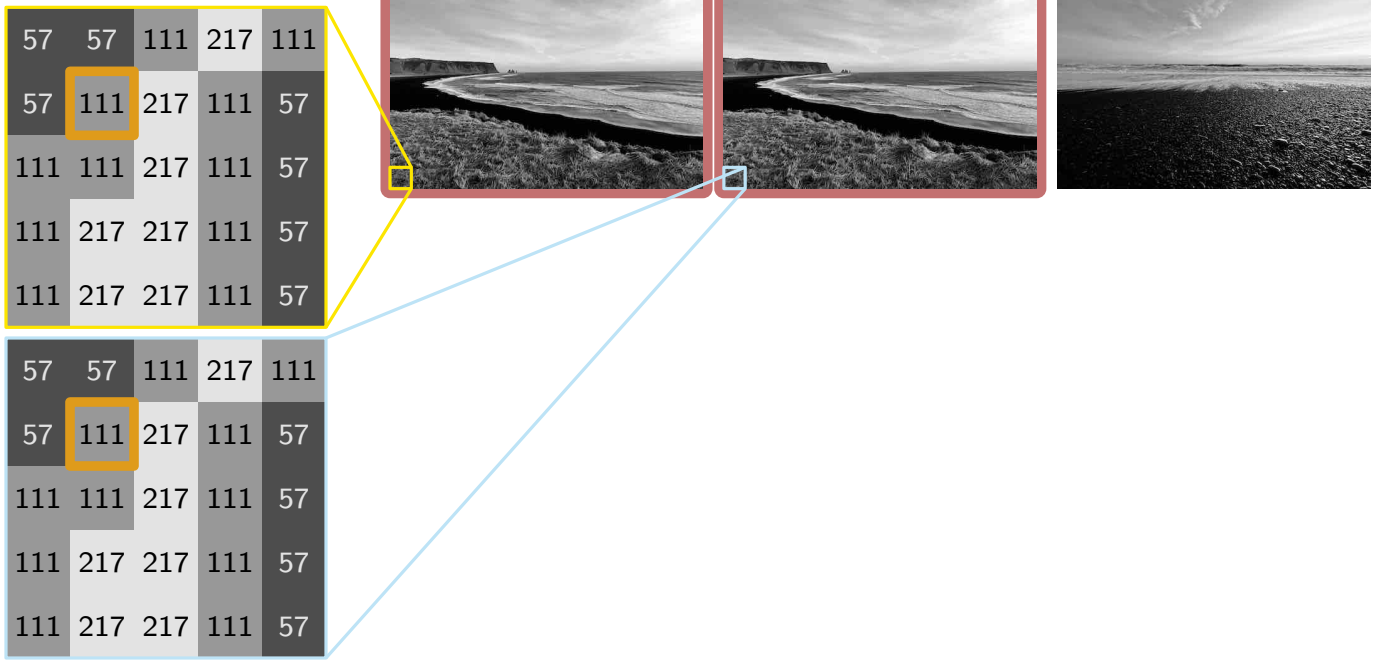




Platz sparen

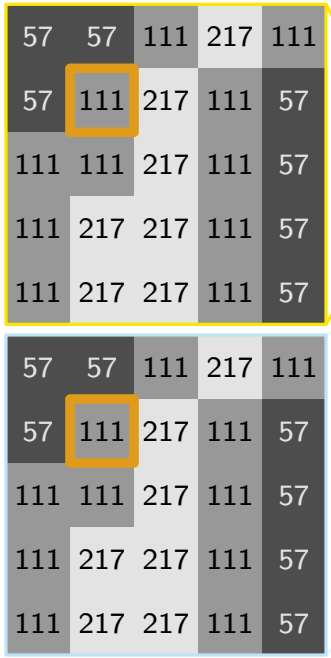
- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit
- Nicht billig, aber nehmen wir in Kauf

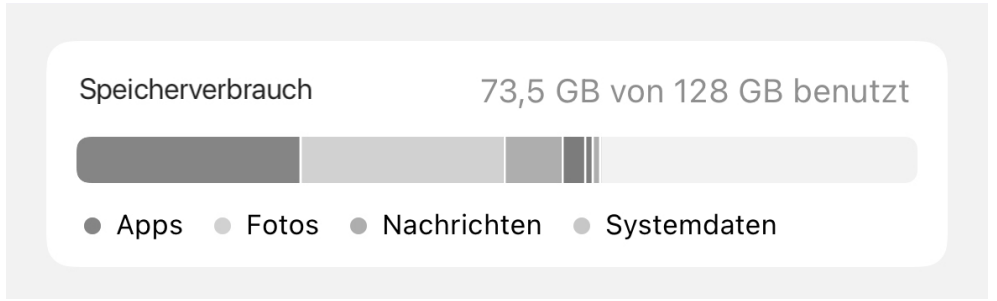




Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit
- Nicht billig, aber nehmen wir in Kauf

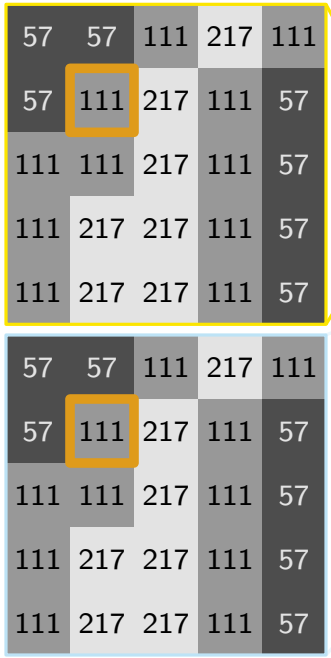


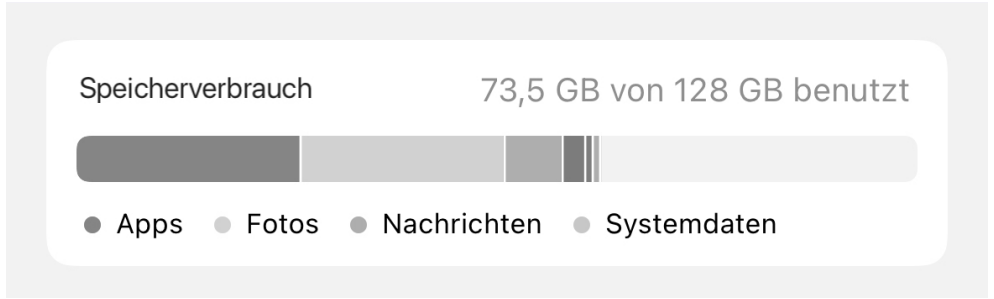


- Sei n die Anzahl der Bilder

Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit
- Nicht billig, aber nehmen wir in Kauf





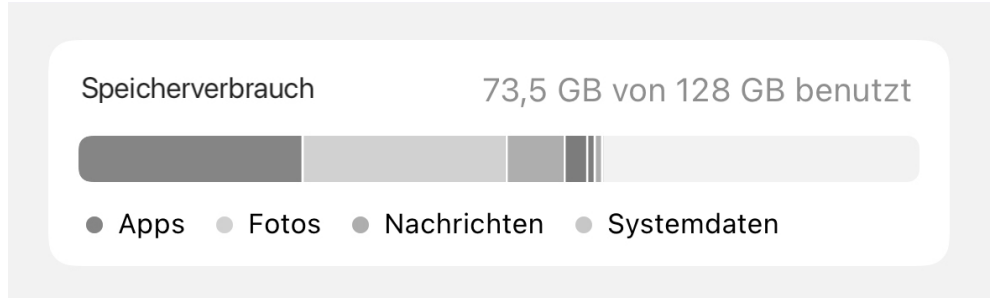
- Sei n die Anzahl der Bilder
- Vergleiche jedes Bild mit jedem anderen

Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit
- Nicht billig, aber nehmen wir in Kauf

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57



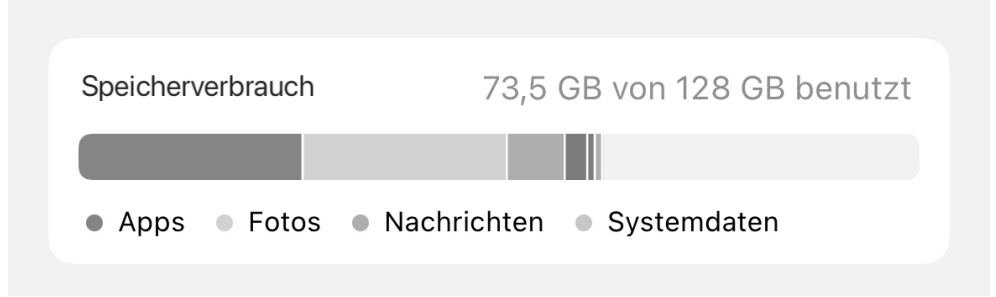
- Sei n die Anzahl der Bilder
- Vergleiche jedes Bild mit jedem anderen
- $\Theta(n^2)$ nicht-billige Vergleiche...

Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit
- Nicht billig, aber nehmen wir in Kauf

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57



- Sei n die Anzahl der Bilder
- Vergleiche jedes Bild mit jedem anderen
- $\Theta(n^2)$ nicht-billige Vergleiche...

Das muss besser gehen...

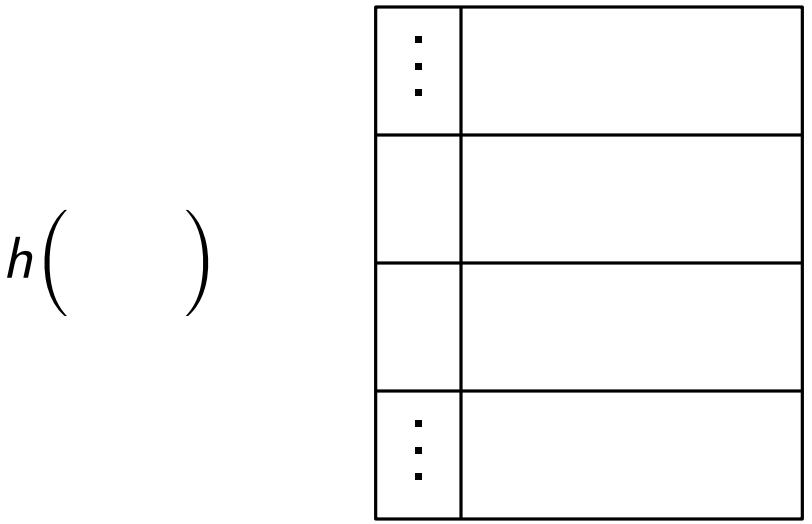
Fotos hashen

- Definiere eine Hashfunktion h für Fotos

$$h\left(\quad\right)$$

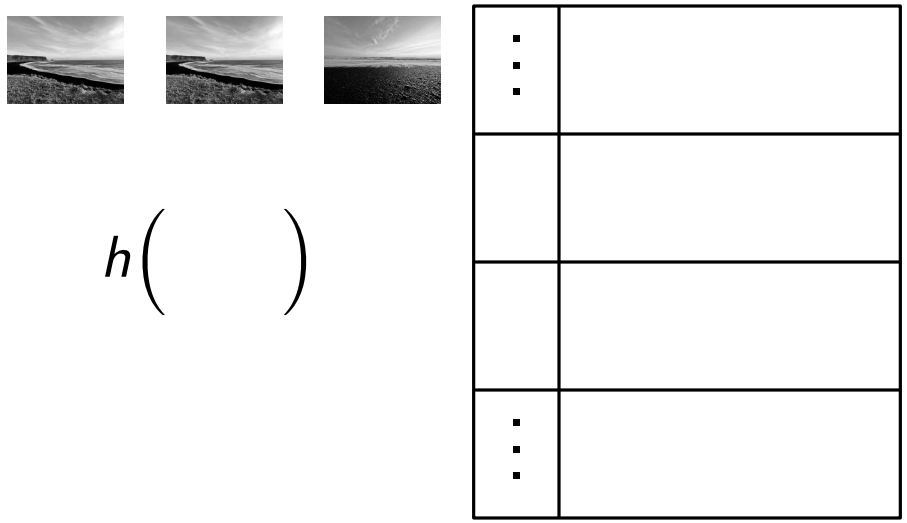
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets



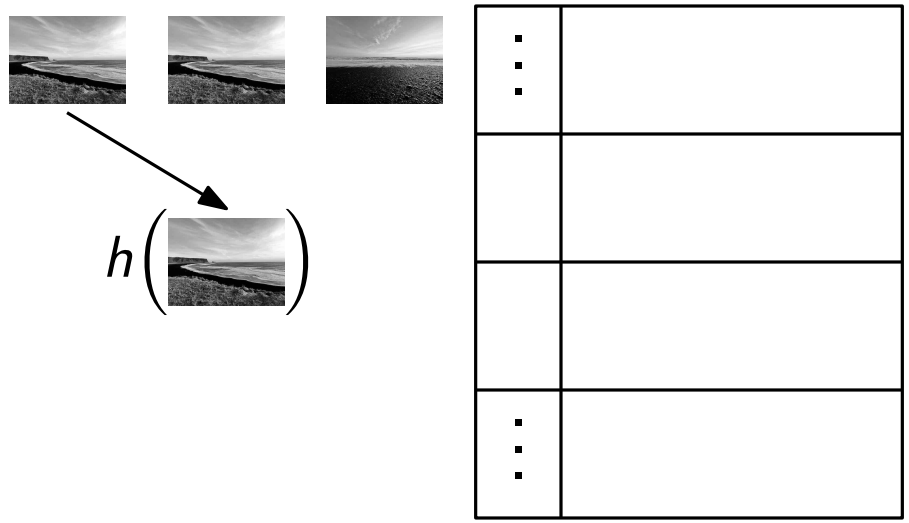
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets



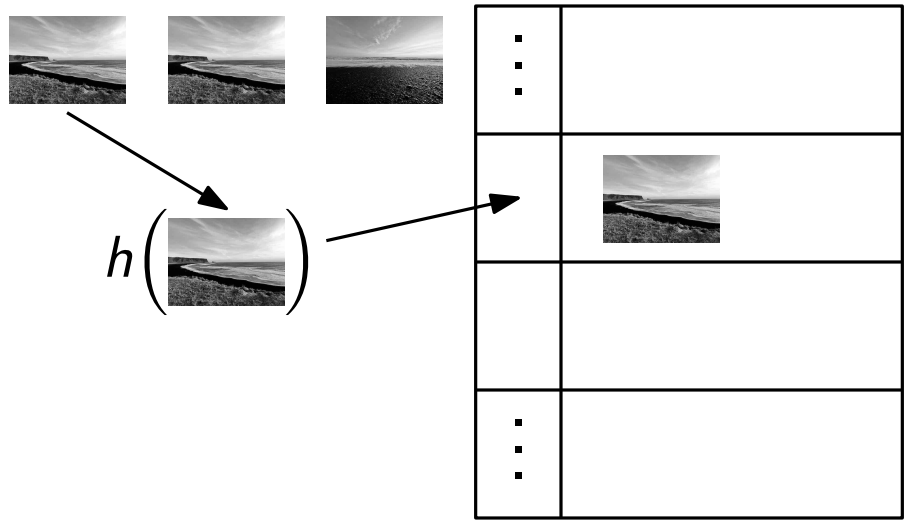
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets



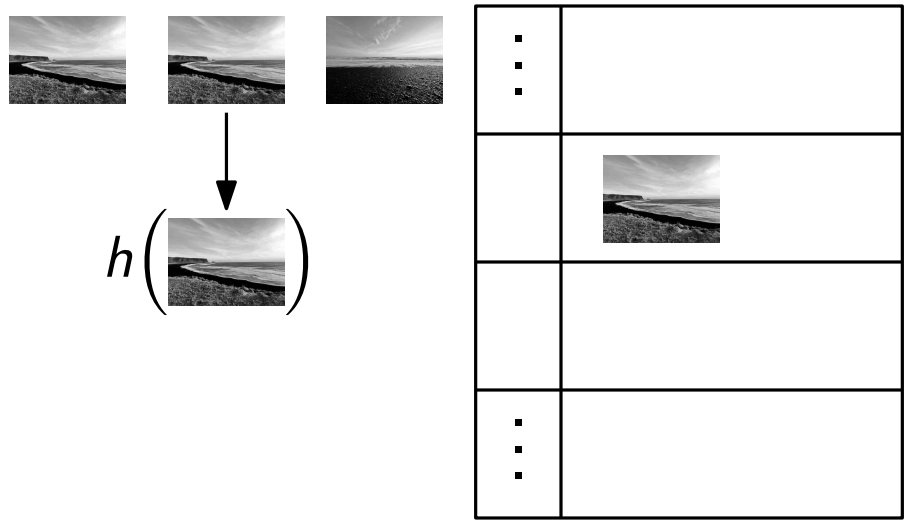
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets



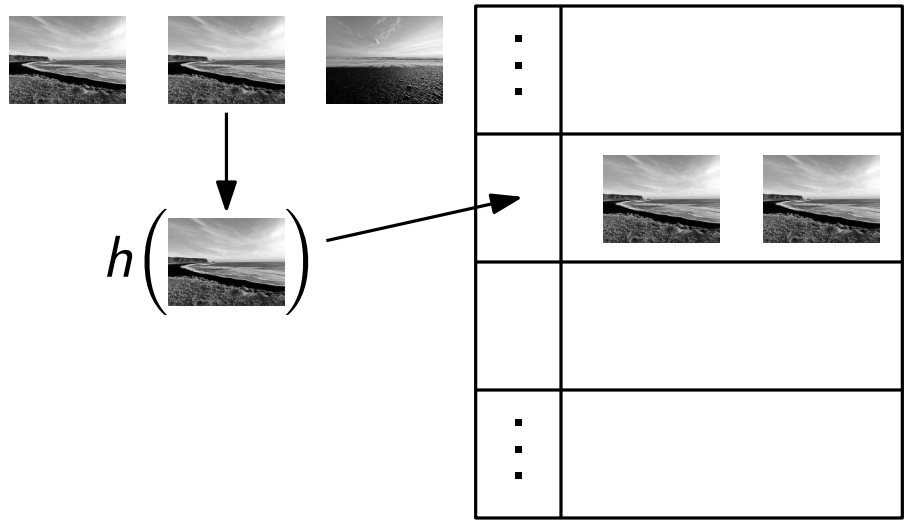
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets



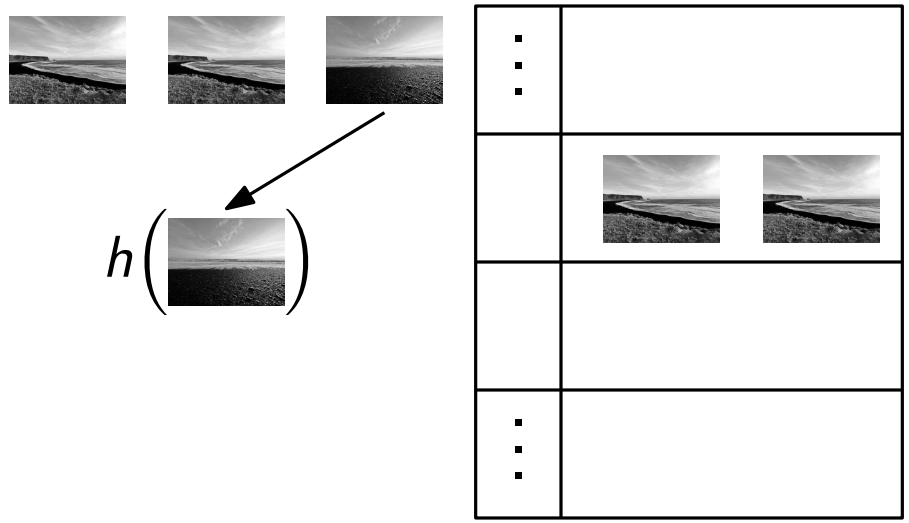
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets



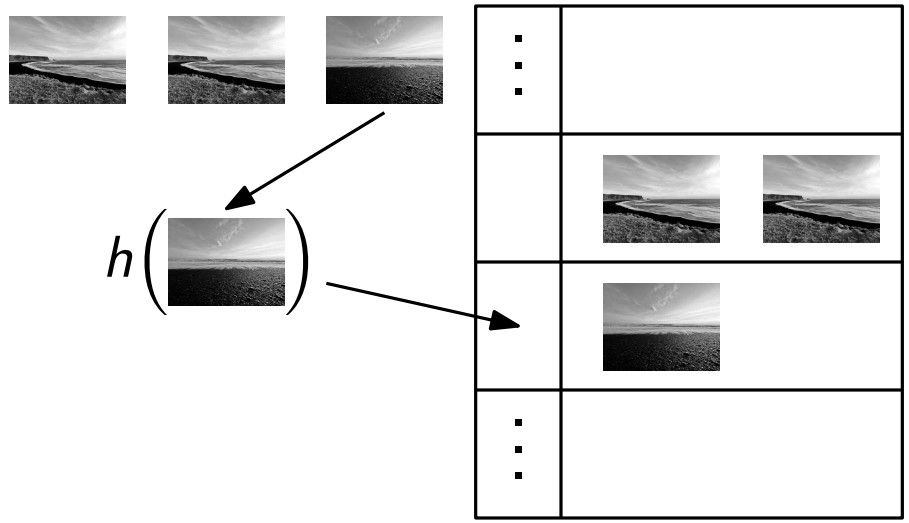
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets



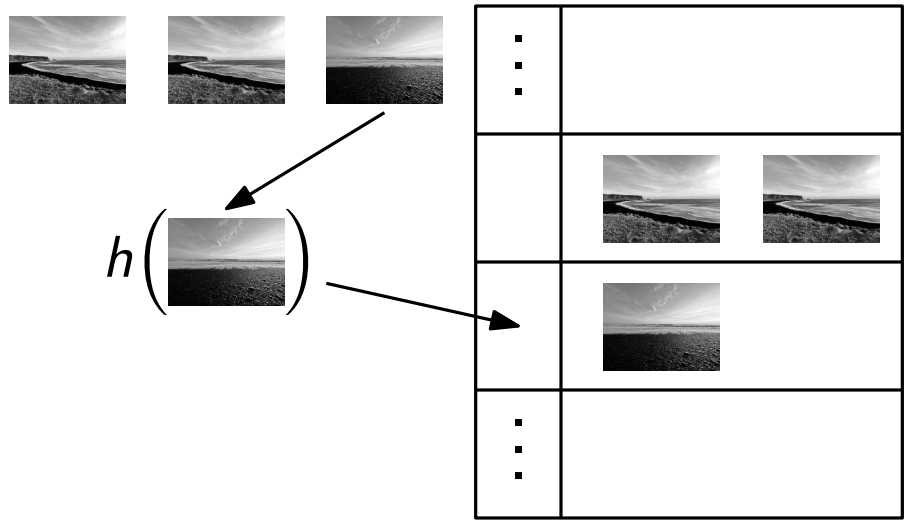
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket



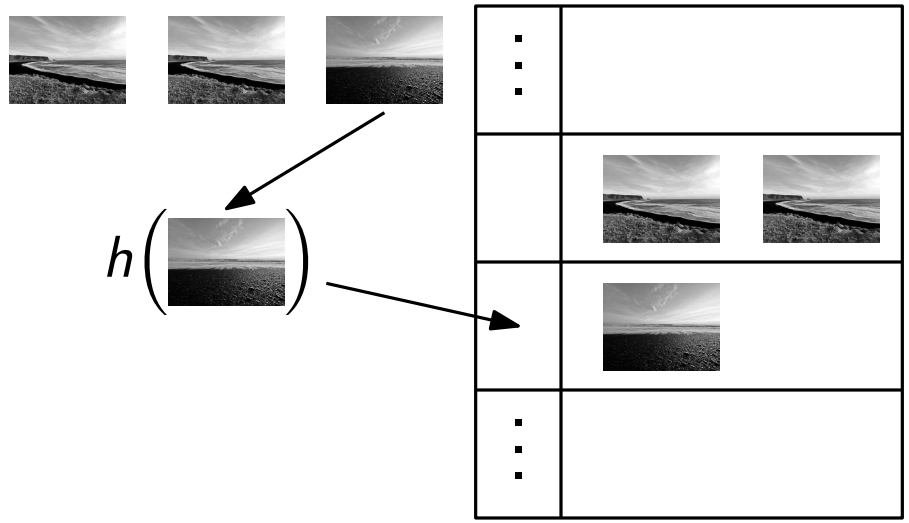
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?



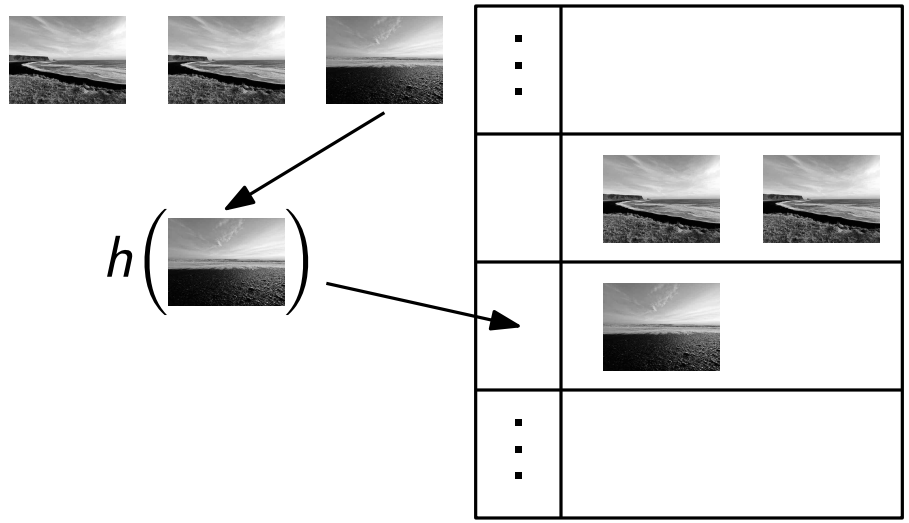
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben



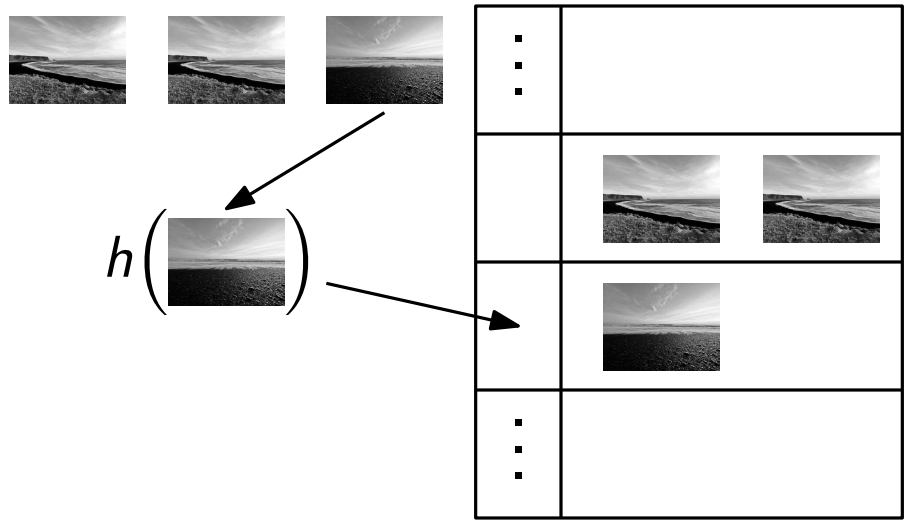
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos



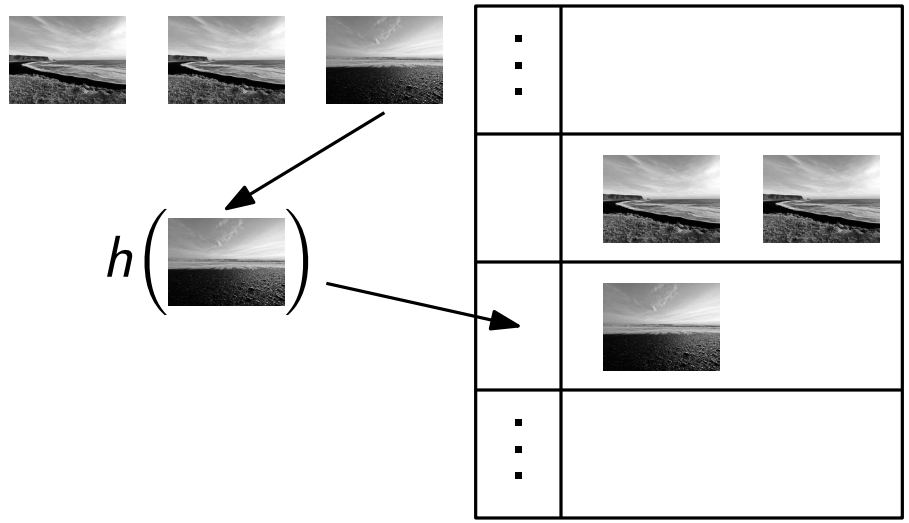
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!



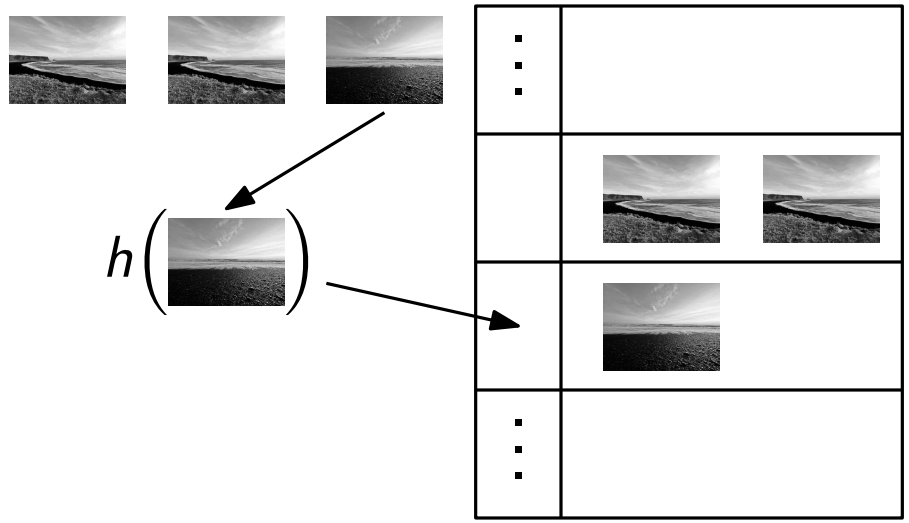
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!
- **Problem**: Wie hashen wir ein Foto?



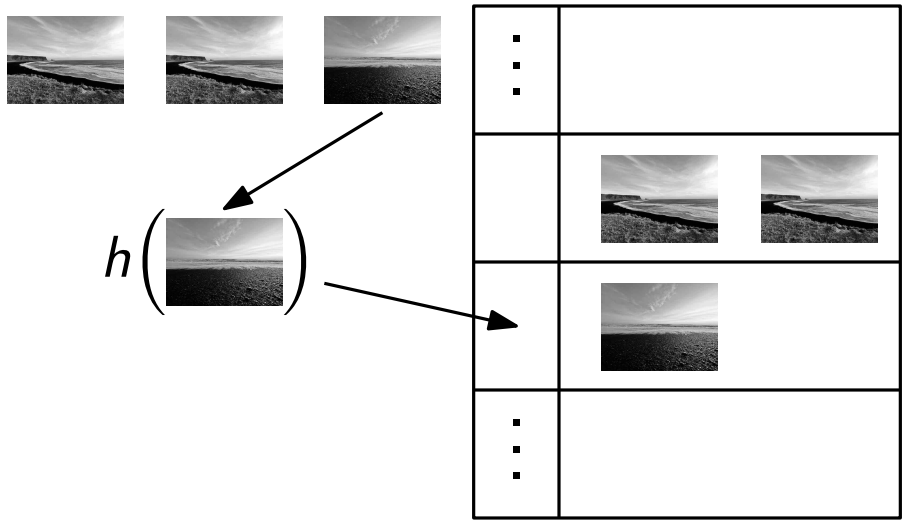
Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!
- **Problem**: Wie hashen wir ein Foto?
- **Idee**: Summe der Pixelwerte



Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!
- **Problem**: Wie hashen wir ein Foto?
- **Idee**: Summe der Pixelwerte

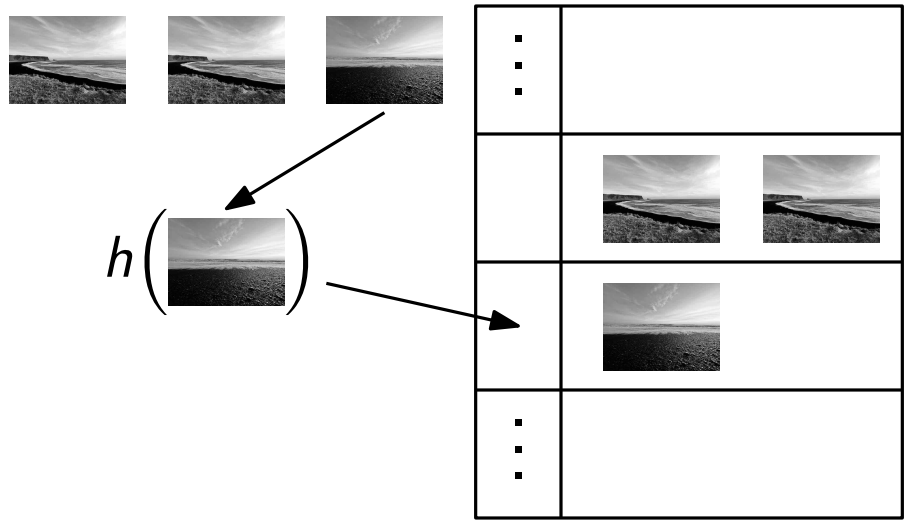


$h($  $) = 57 + 57 + 111 + 217 + \ldots$

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!
- **Problem**: Wie hashen wir ein Foto?
- **Idee**: Summe der Pixelwerte
  - zwei Mal das gleiche Foto gibt den gleichen Wert

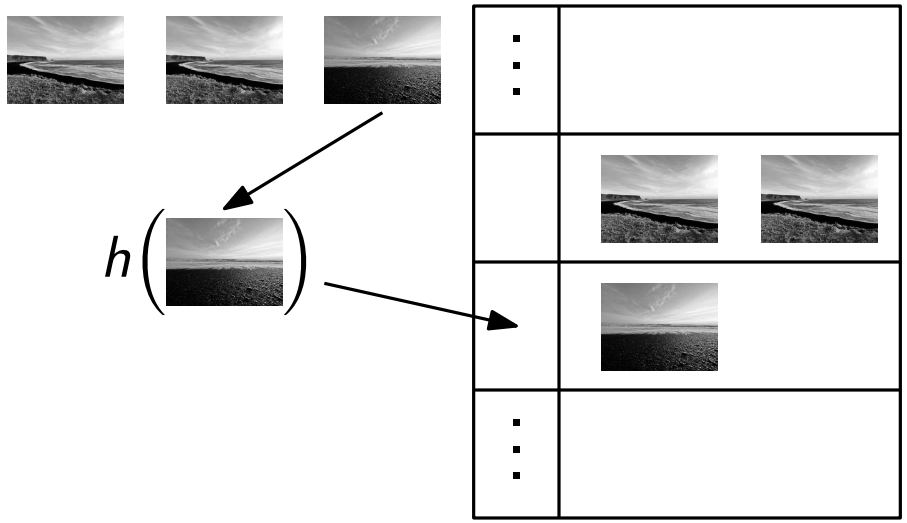


$h($  $) = 57 + 57 + 111 + 217 + \ldots$

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!
- **Problem**: Wie hashen wir ein Foto?
- **Idee**: Summe der Pixelwerte
  - zwei Mal das gleiche Foto gibt den gleichen Wert
  - Duplikate landen im gleichen Bucket

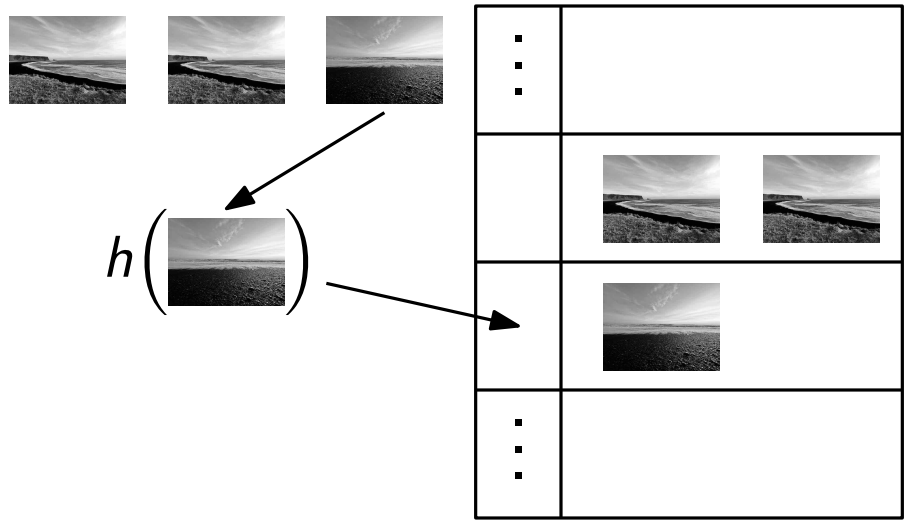


$h($  $) = 57 + 57 + 111 + 217 + \ldots$

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!
- **Problem**: Wie hashen wir ein Foto?
- **Idee**: Summe der Pixelwerte
  - zwei Mal das gleiche Foto gibt den gleichen Wert
  - Duplikate landen im gleichen Bucket
  - h bildet auf große Wert ab $\rightarrow$ benutze zweite Hashfunktion um Indizes zu erhalten



$h\left(\text{  }\right) = 57 + 57 + 111 + 217 + \ldots$

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ...

Fotos hashen

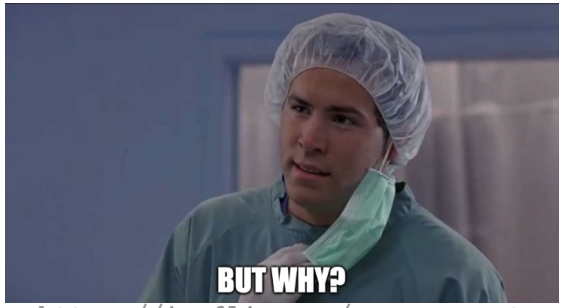
https://youtu.be/r-TLSBdHe1A?t=2239

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



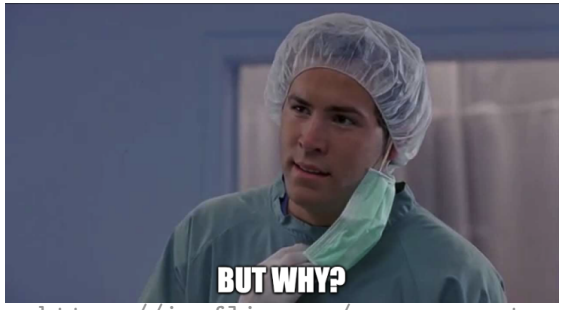
https://imgflip.com/memegenerator

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

Exkurs: Wahrscheinlichkeitstheorie

https://youtu.be/r-TLSBdHe1A?t=2239

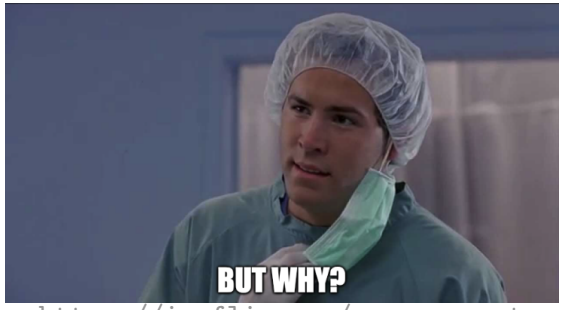


https://imgflip.com/memegenerator

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



https://imgflip.com/memegenerator

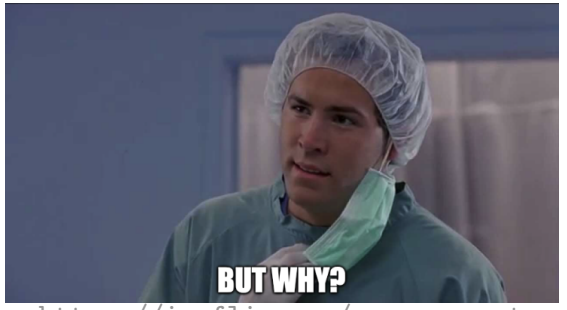
Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



https://imgflip.com/memegenerator

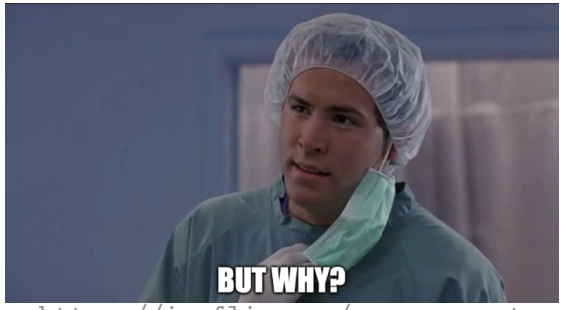
Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



https://imgflip.com/memegenerator

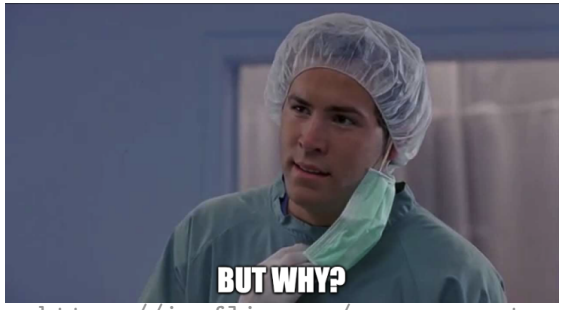
Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



https://imgflip.com/memegenerator

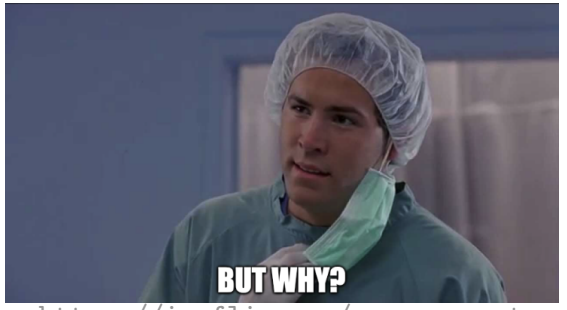
Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



https://imgflip.com/memegenerator

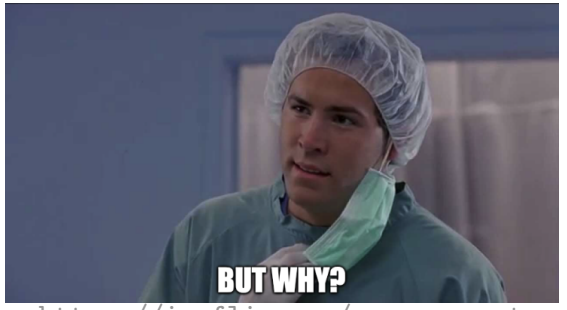
Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

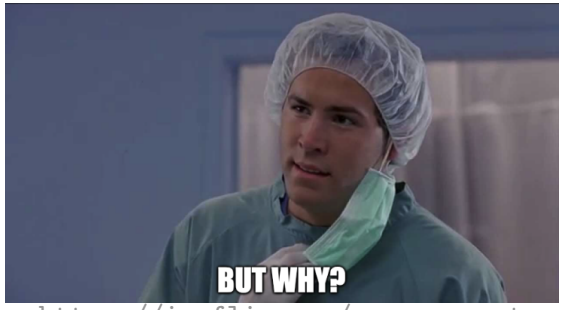
- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

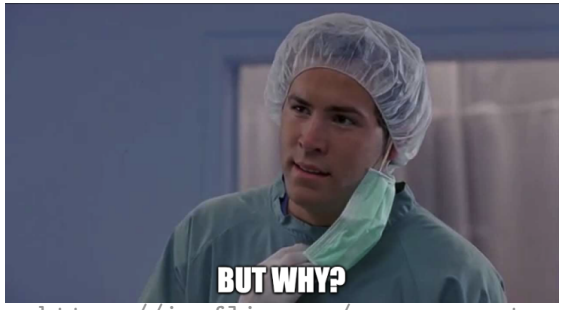
$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



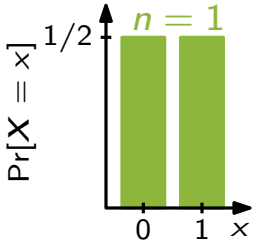
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

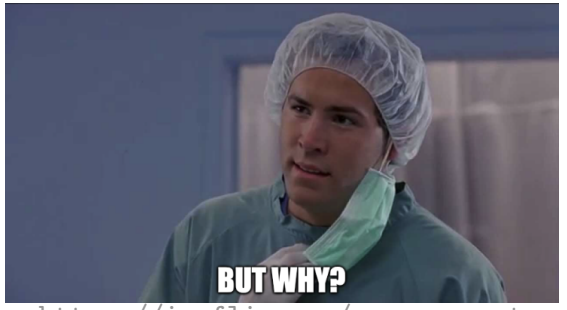
Verteilung von X



Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



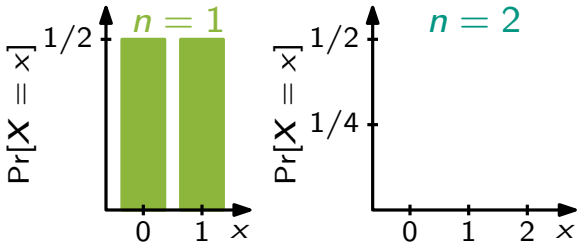
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

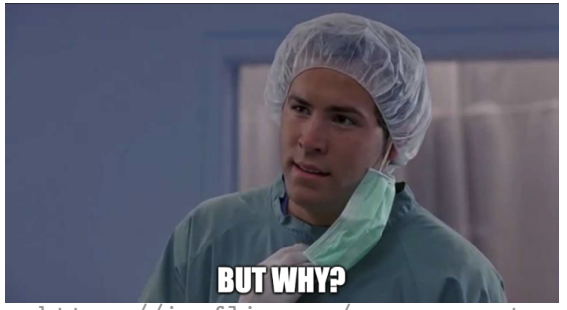
Verteilung von X



Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



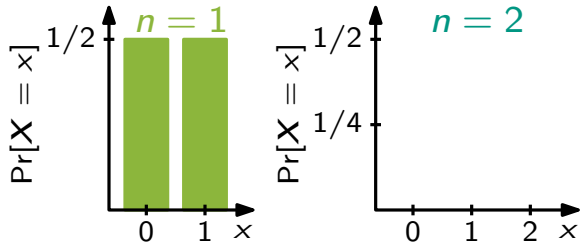
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X

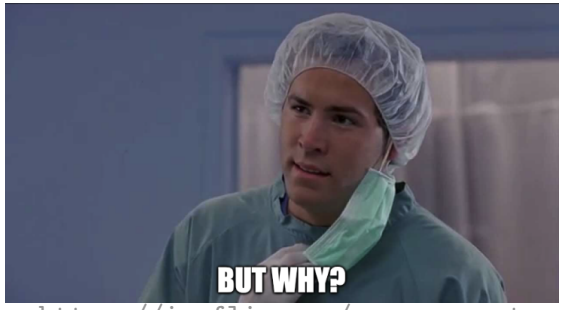


$X_1 = 0, X_2 = 0 \Rightarrow X = 0$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



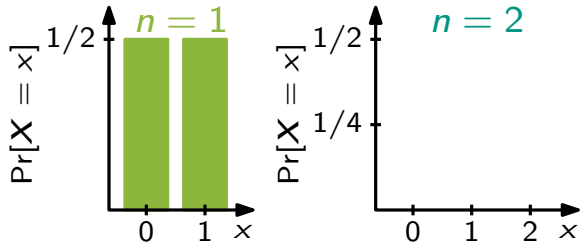
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



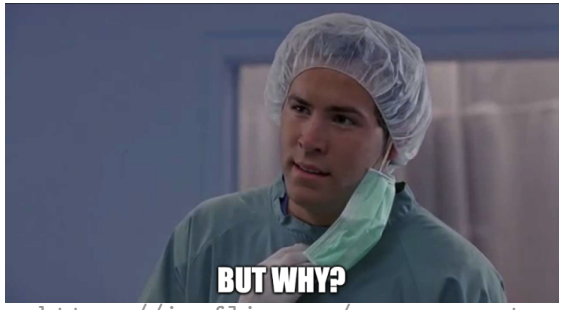
$X_1 = 0, X_2 = 0 \Rightarrow X = 0$

$X_1 = 0, X_2 = 1 \Rightarrow X = 1$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



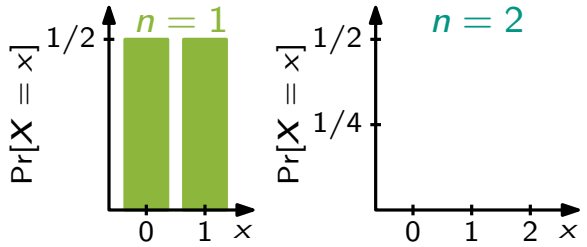
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



$X_1 = 0, X_2 = 0 \Rightarrow X = 0$

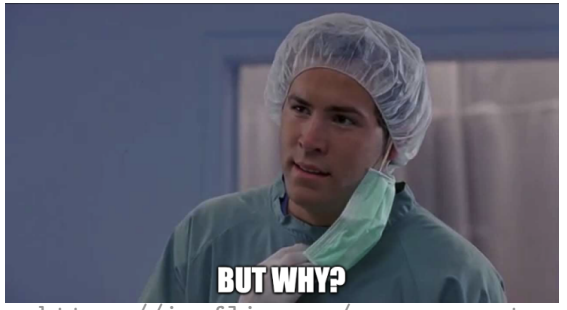
$X_1 = 0, X_2 = 1 \Rightarrow X = 1$

$X_1 = 1, X_2 = 0 \Rightarrow X = 1$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



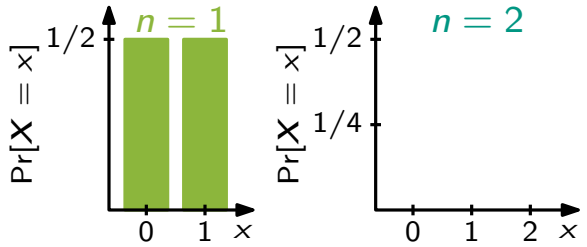
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



$X_1 = 0, X_2 = 0 \Rightarrow X = 0$

$X_1 = 0, X_2 = 1 \Rightarrow X = 1$

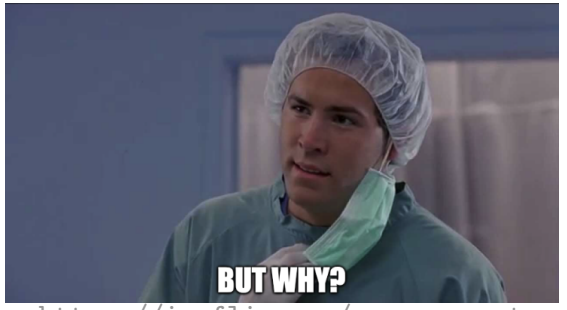
$X_1 = 1, X_2 = 0 \Rightarrow X = 1$

$X_1 = 1, X_2 = 1 \Rightarrow X = 2$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



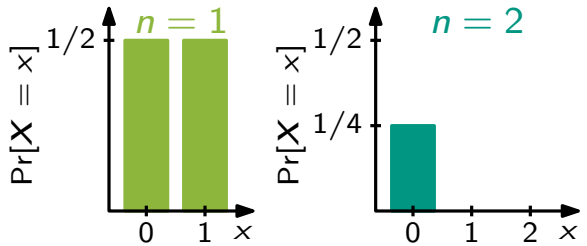
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



$X_1 = 0, X_2 = 0 \Rightarrow X = 0$

$X_1 = 0, X_2 = 1 \Rightarrow X = 1$

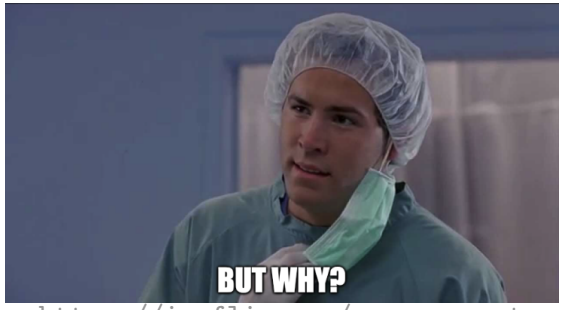
$X_1 = 1, X_2 = 0 \Rightarrow X = 1$

$X_1 = 1, X_2 = 1 \Rightarrow X = 2$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



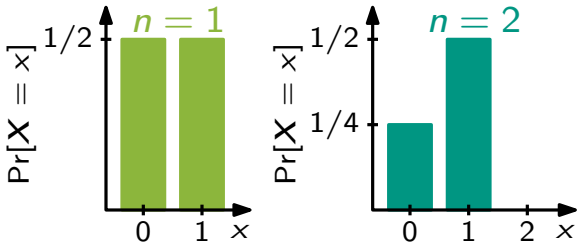
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



$X_1 = 0, X_2 = 0 \Rightarrow X = 0$

$X_1 = 0, X_2 = 1 \Rightarrow X = 1$

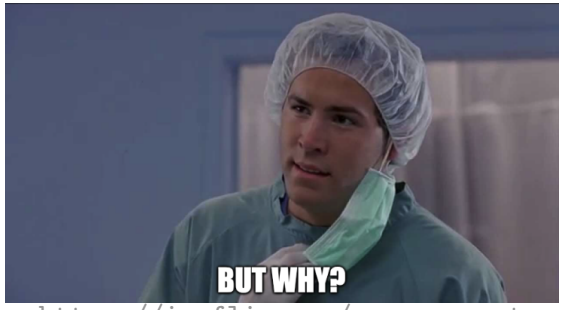
$X_1 = 1, X_2 = 0 \Rightarrow X = 1$

$X_1 = 1, X_2 = 1 \Rightarrow X = 2$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



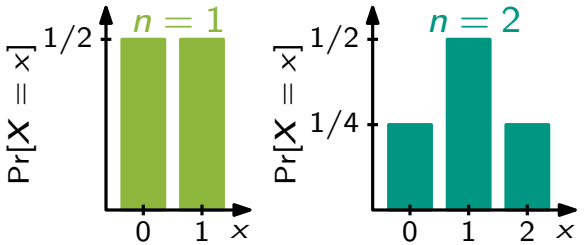
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



$X_1 = 0, X_2 = 0 \Rightarrow X = 0$

$X_1 = 0, X_2 = 1 \Rightarrow X = 1$

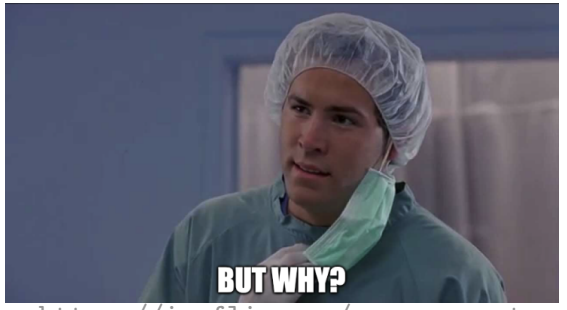
$X_1 = 1, X_2 = 0 \Rightarrow X = 1$

$X_1 = 1, X_2 = 1 \Rightarrow X = 2$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



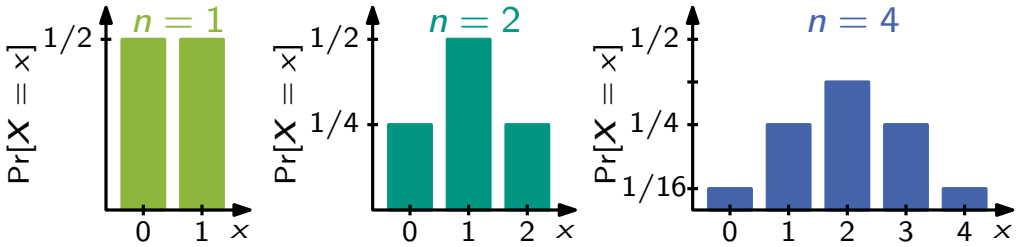
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

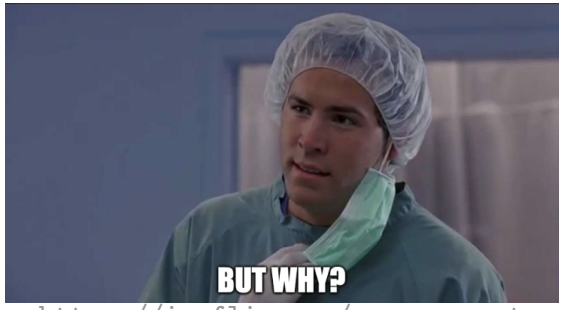
Verteilung von X



Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



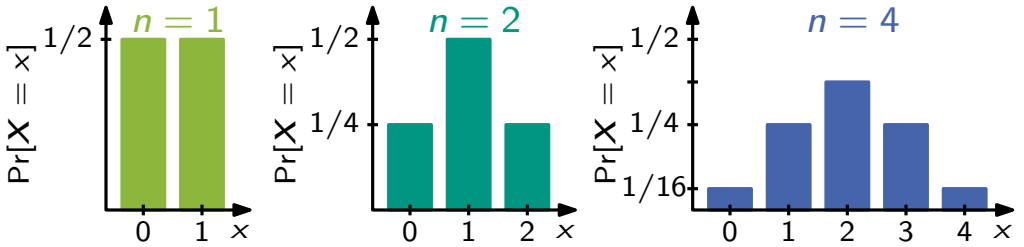
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X

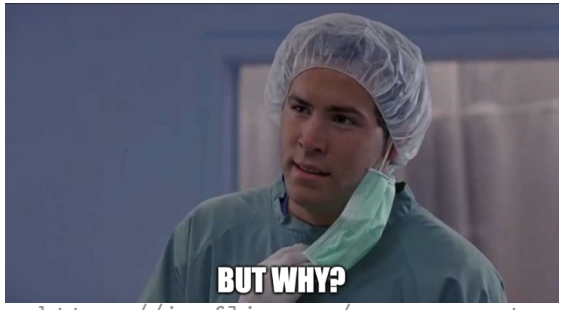


$\Pr[X = x] =$

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



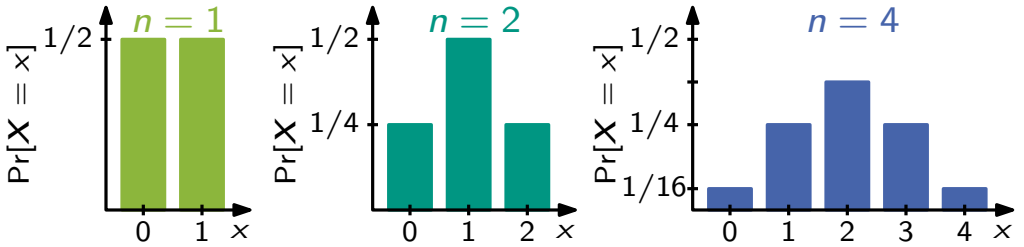
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



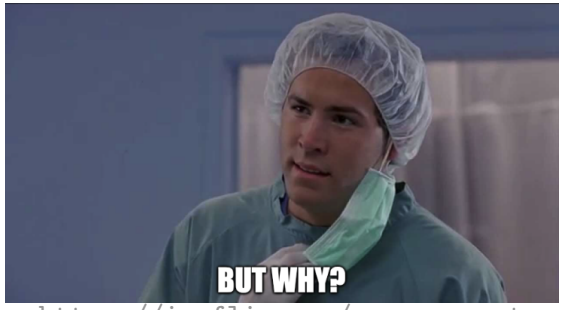
$$\Pr[X = x] = \binom{n}{?} \frac{1}{2^n}$$

Wie viele Bit-Strings der Länge n gibt es?

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



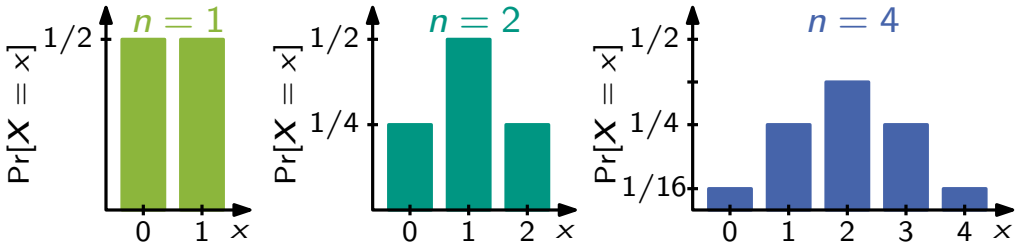
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



Wie viele Bit-Strings der Länge n gibt es die x mal eine 1 enthalten?

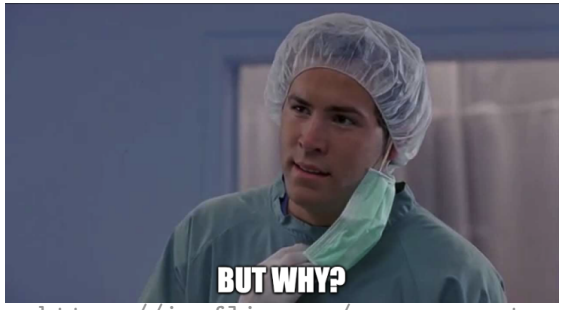
$$\Pr[X = x] = \frac{\binom{n}{x}}{2^n}$$

Wie viele Bit-Strings der Länge n gibt es?

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



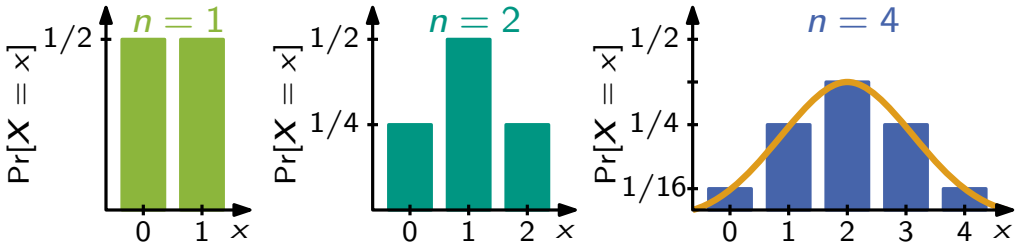
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



Wie viele Bit-Strings der Länge n gibt es die x mal eine 1 enthalten?

$$\Pr[X = x] = \frac{\binom{n}{x}}{2^n}$$

Wie viele Bit-Strings der Länge n gibt es?

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$

Exkurs: Wahrscheinlichkeitstheorie

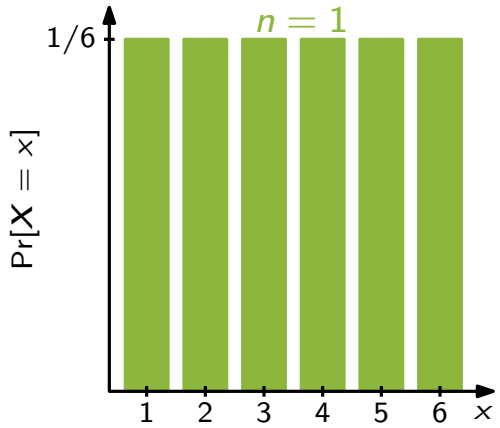
- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?

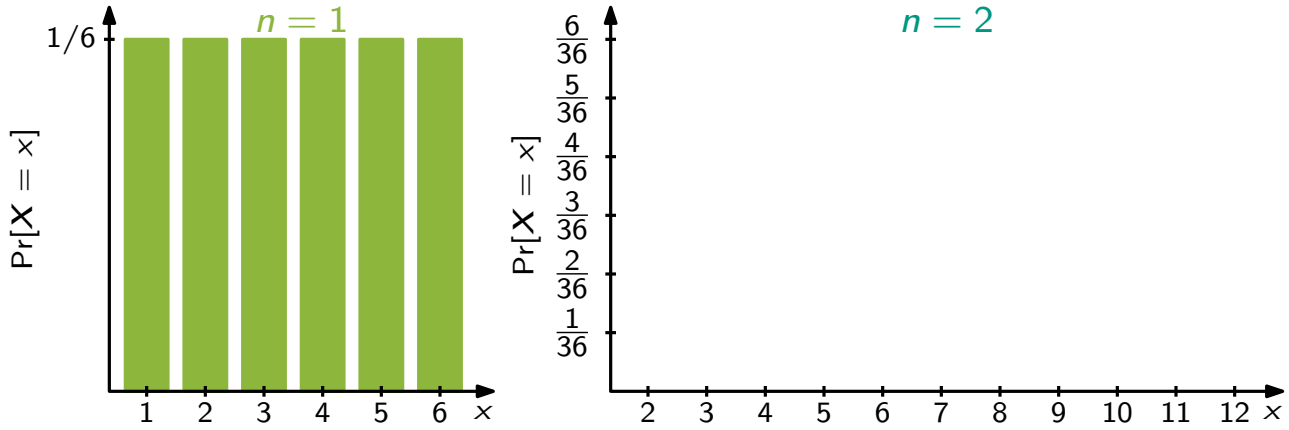
Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



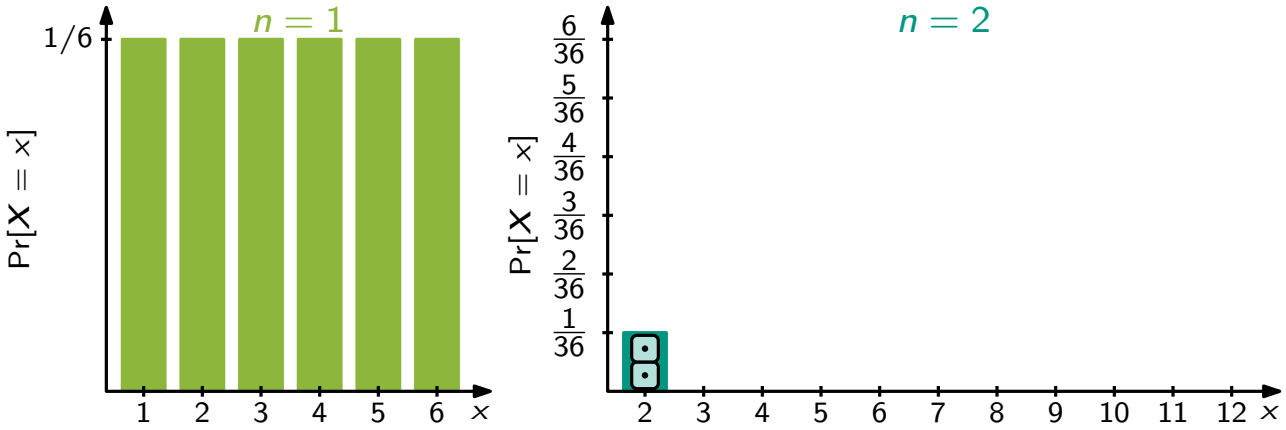
Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



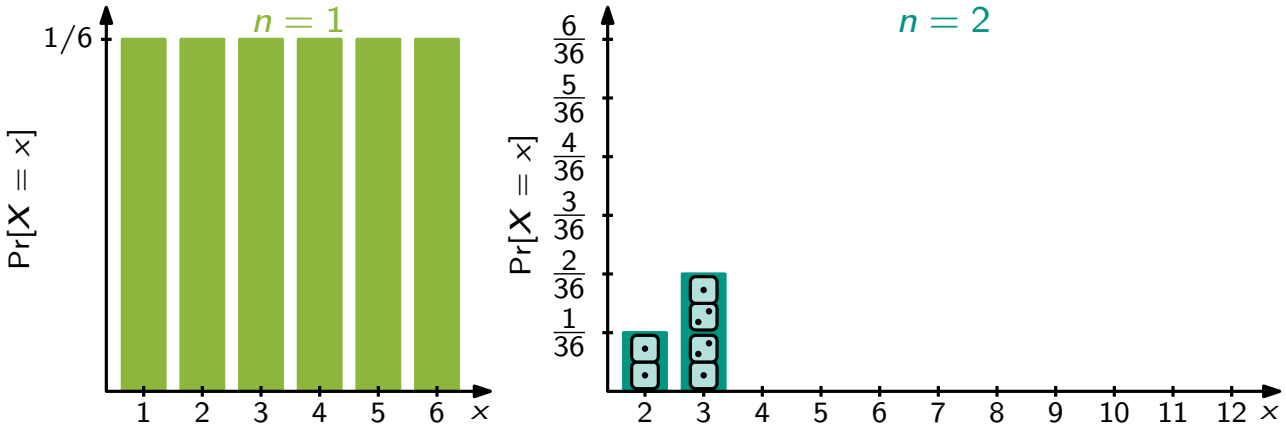
Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



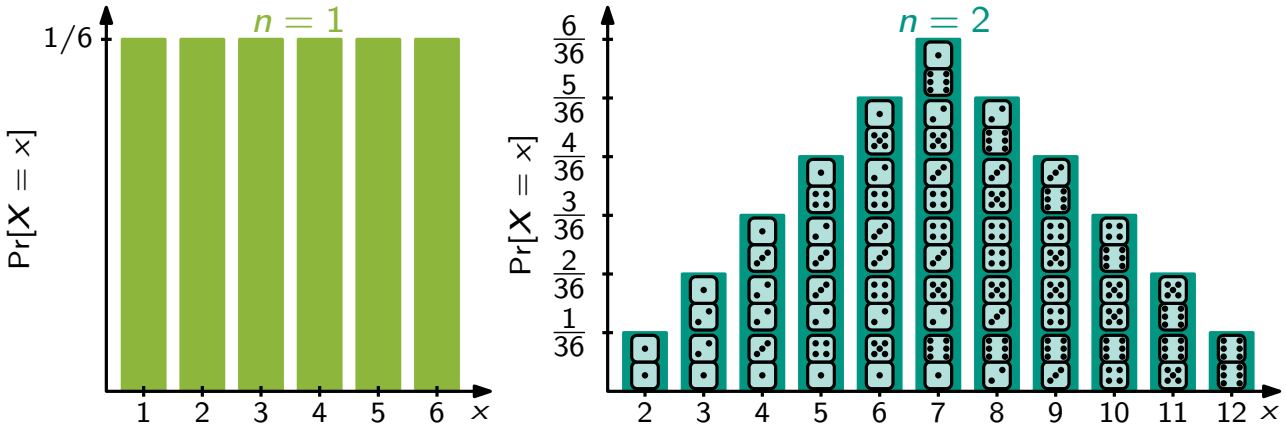
Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



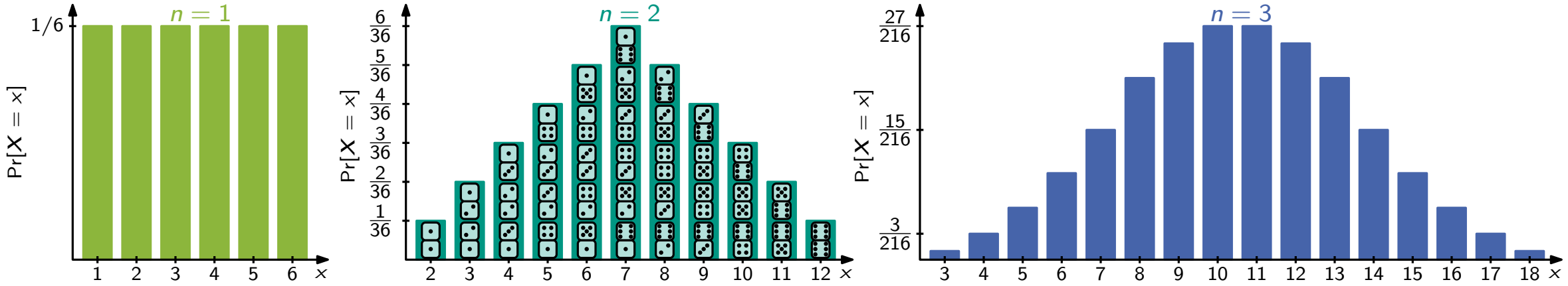
Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



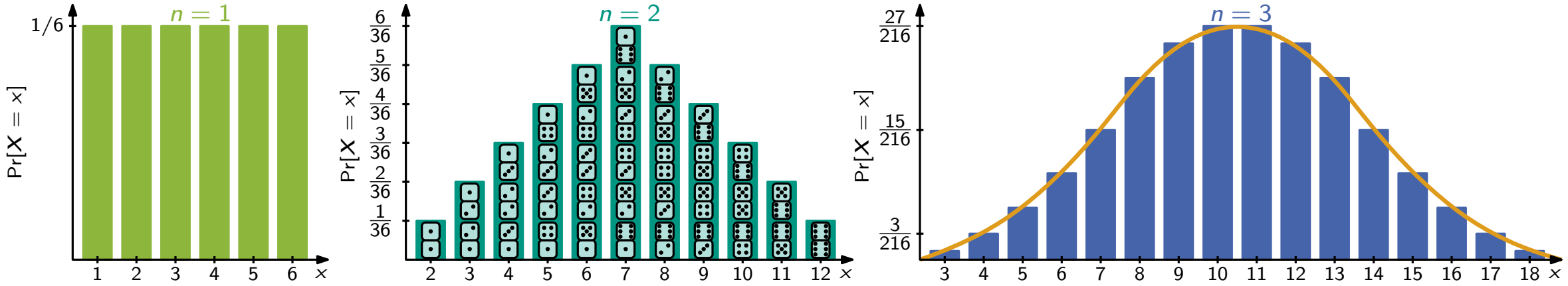
Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



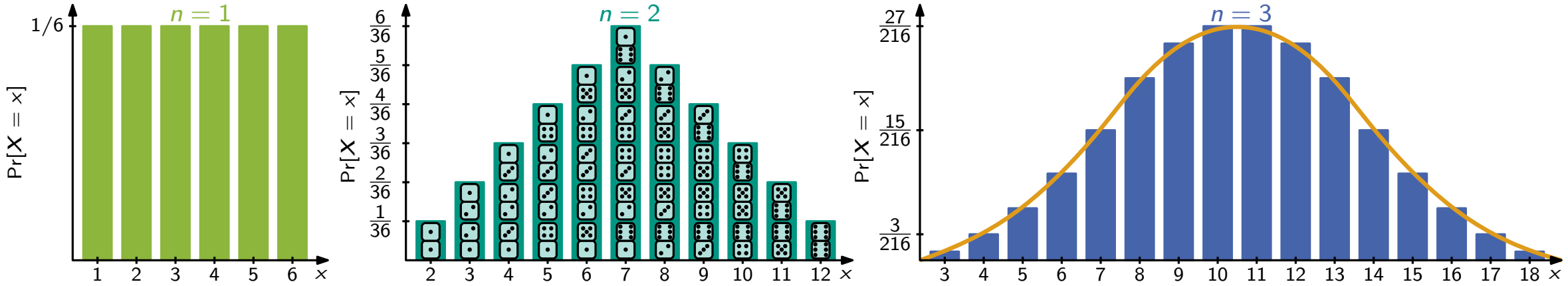
Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



Exkurs: Wahrscheinlichkeitstheorie

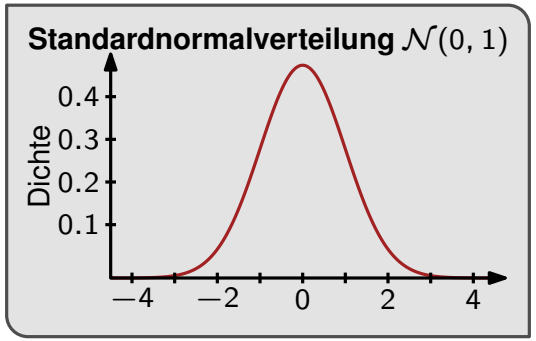
- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?

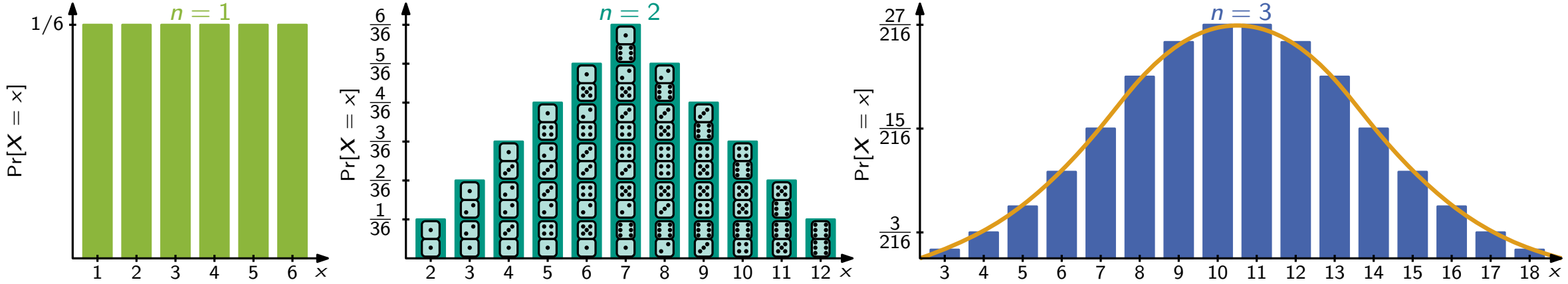


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0, 1)$.

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



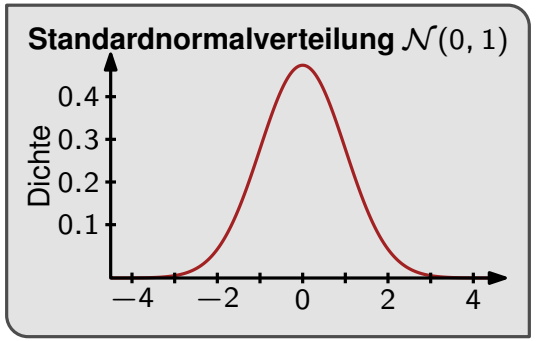


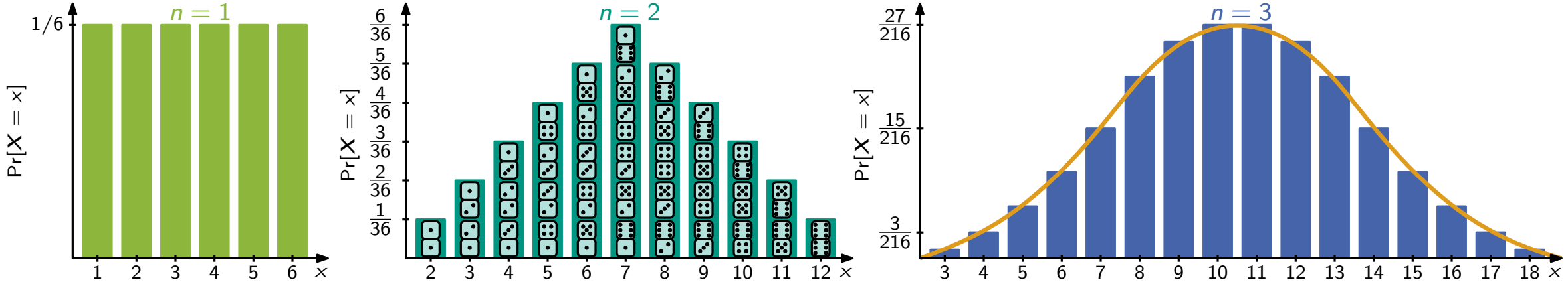
Zentraler Grenzwertsatz

Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?



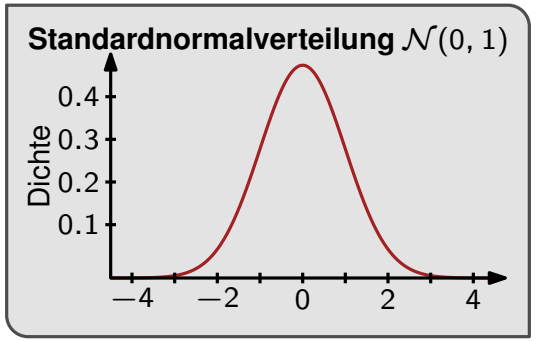


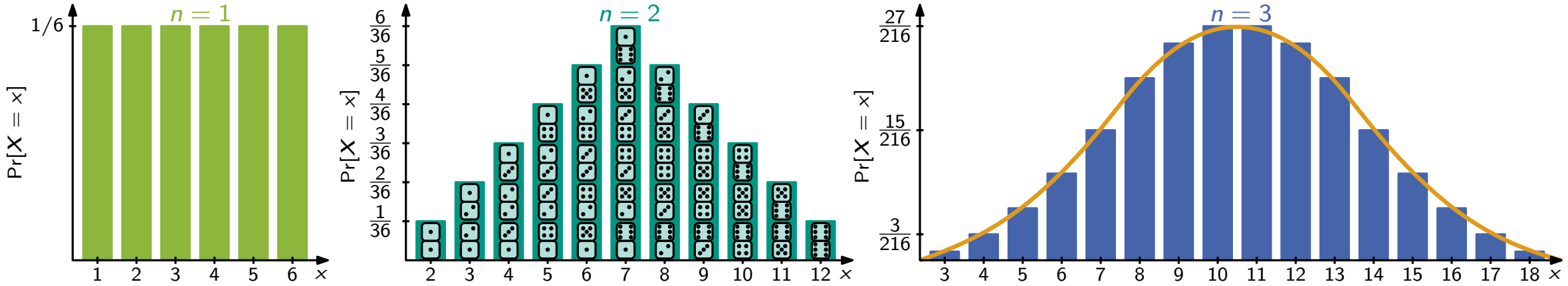
Zentraler Grenzwertsatz

Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?





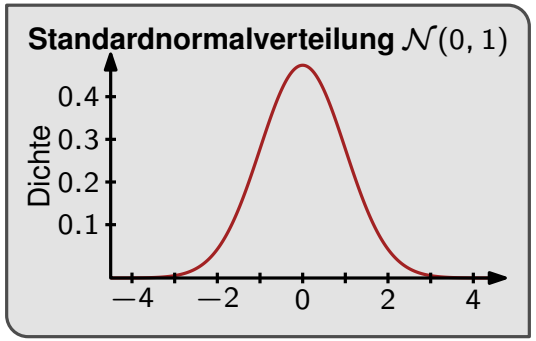
Zentraler Grenzwertsatz

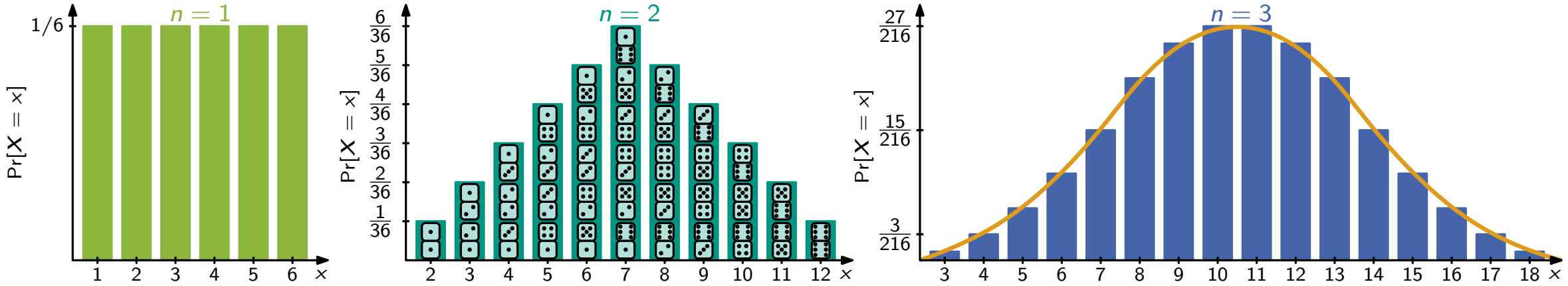
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

verschieben

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?





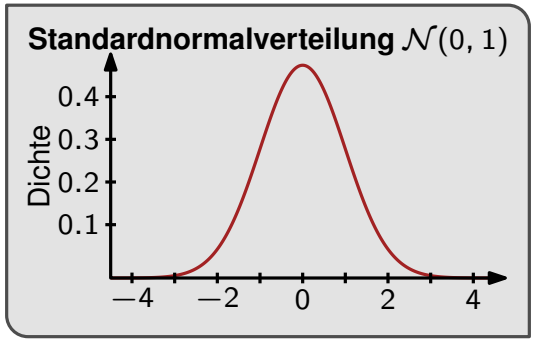
Zentraler Grenzwertsatz

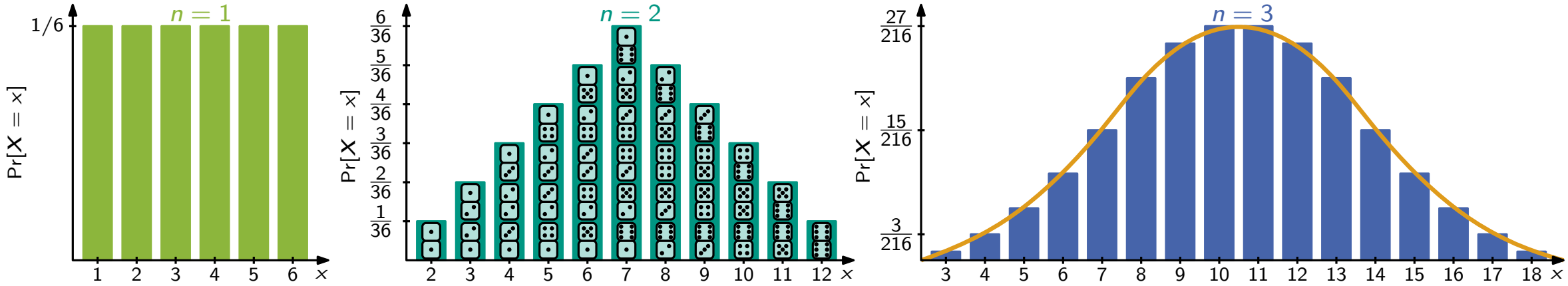
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

verschieben stauchen

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?





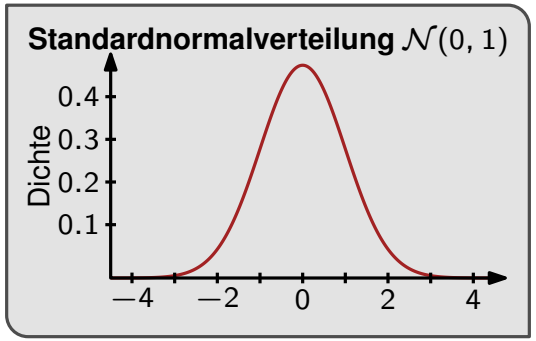
Zentraler Grenzwertsatz Die ursprüngliche Verteilung spielt keine Rolle!

Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

verschieben stauchen

Fotos hashen

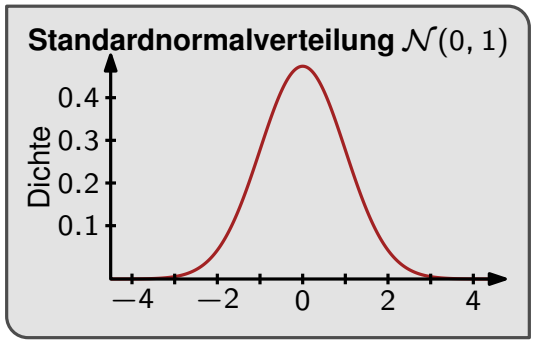
- Was hat das mit unseren Fotos zu tun?



Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

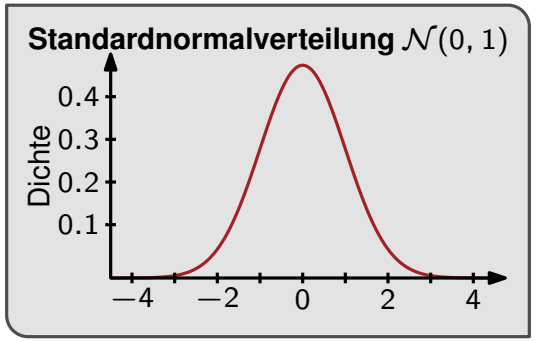
- Was hat das mit unseren Fotos zu tun?



Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

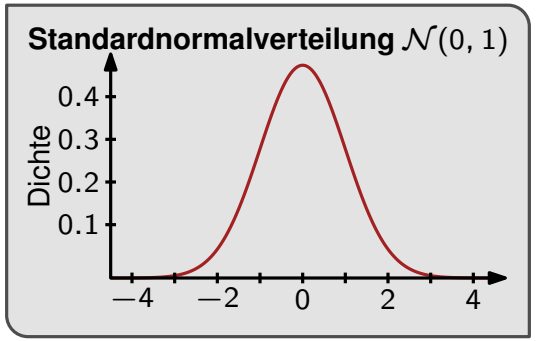
- Was hat das mit unseren Fotos zu tun?



Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

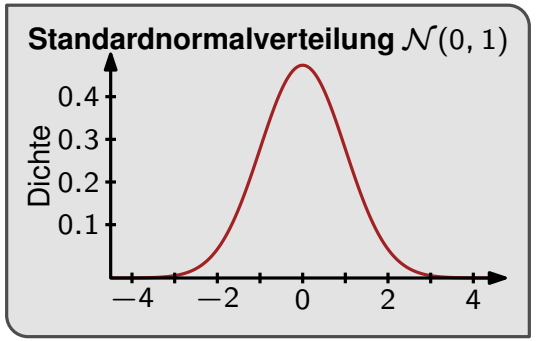
- Was hat das mit unseren Fotos zu tun?



Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

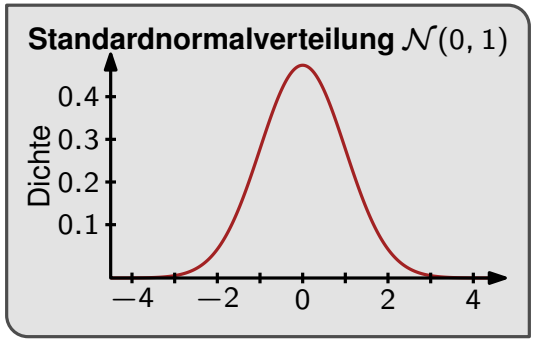
- Was hat das mit unseren Fotos zu tun?



Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

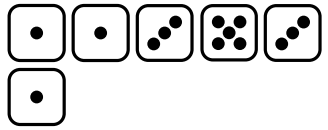
- Was hat das mit unseren Fotos zu tun?

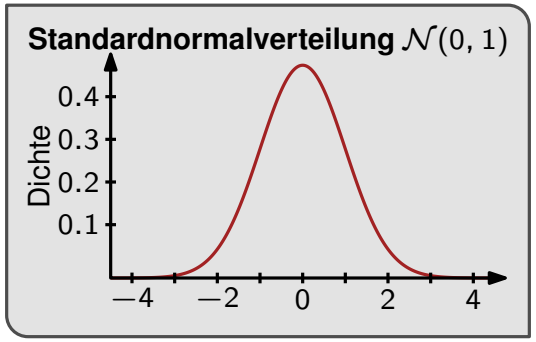


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?

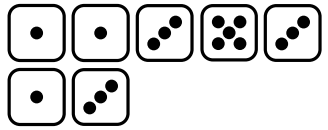


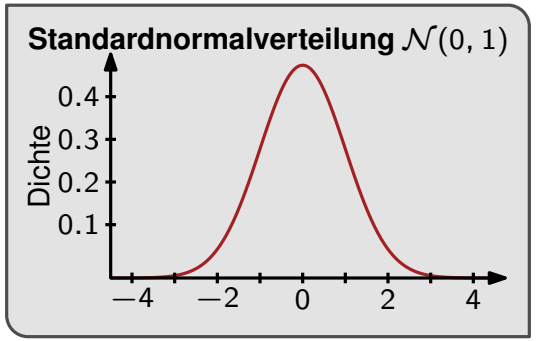


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?

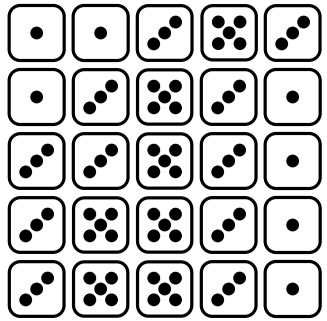


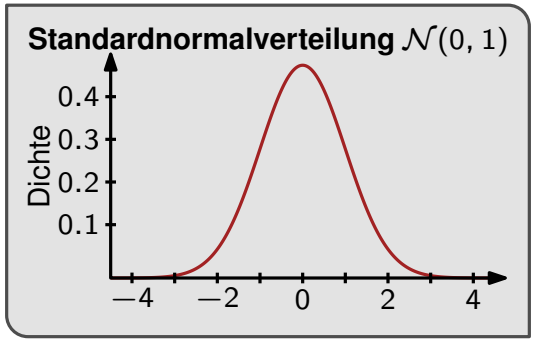


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?

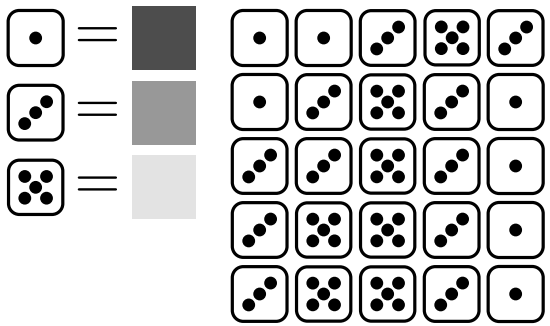


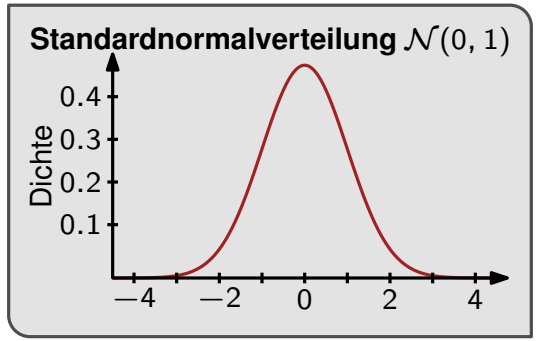


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?

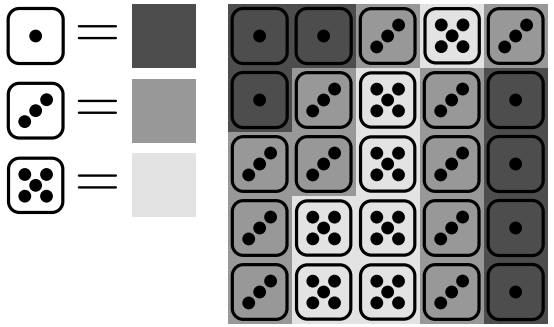


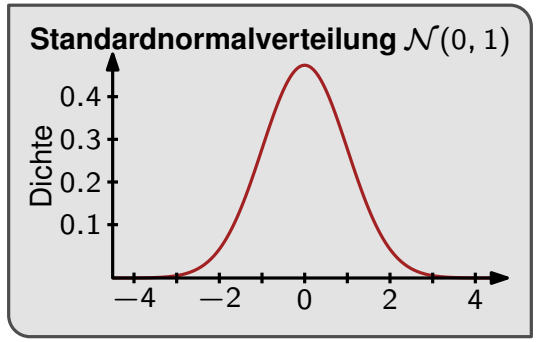


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung

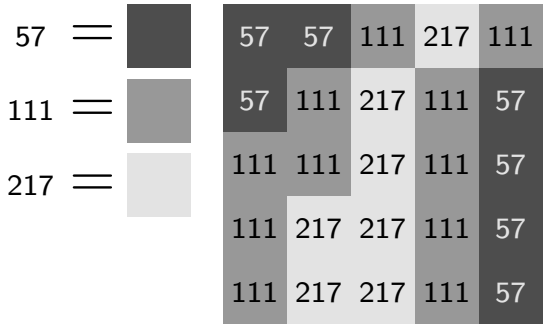


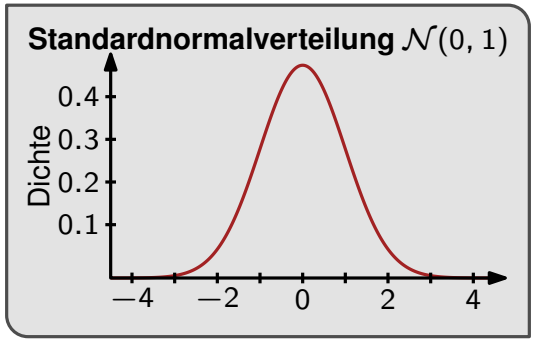


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung
- Unsere Pixel sind 256-seitige Würfel
  (wenn auch vermutlich nicht gleich verteilt)

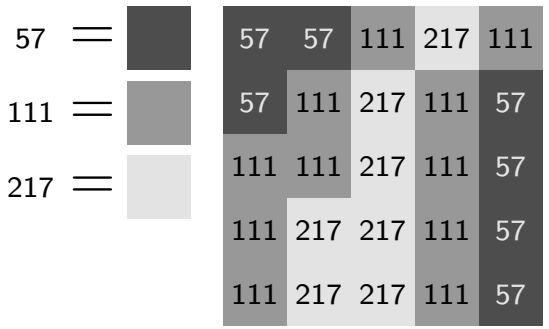


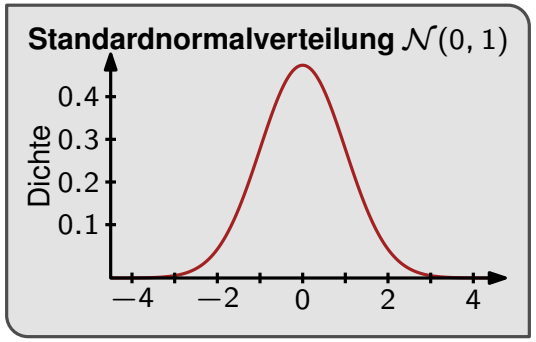


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung
- Unsere Pixel sind 256-seitige Würfel
  (wenn auch vermutlich nicht gleich verteilt)
- Unsere Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet nun die großen Zahlen auf Buckets ab

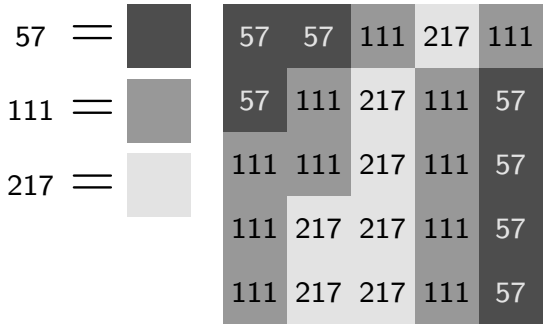


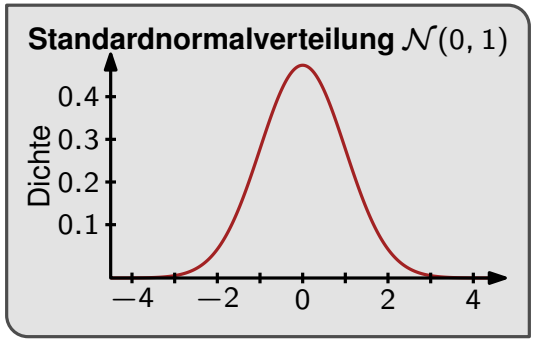


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung
- Unsere Pixel sind 256-seitige Würfel
  (wenn auch vermutlich nicht gleich verteilt)
- Unsere Hashunfktionen
  - die erste bildet Fotos auf ~~große~~ Zahlen ab
    (viele: Fotos; die gleichen: Zahlen)
  - die zweite bildet nun die großen Zahlen auf Buckets ab

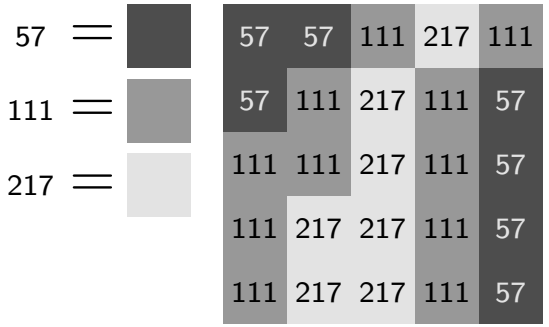


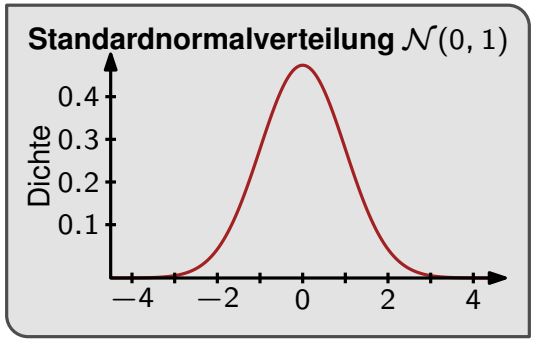


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung
- Unsere Pixel sind 256-seitige Würfel
  (wenn auch vermutlich nicht gleich verteilt)
- Unsere Hashunfktionen
  - die erste bildet Fotos auf ~~große~~ Zahlen ab
    (viele / die gleichen)
  - die zweite bildet nun die ~~großen~~ Zahlen auf Buckets ab
    (die gleichen)

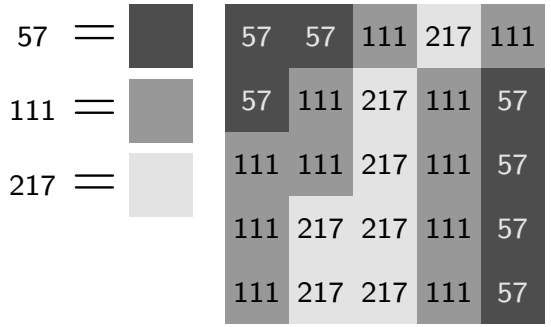


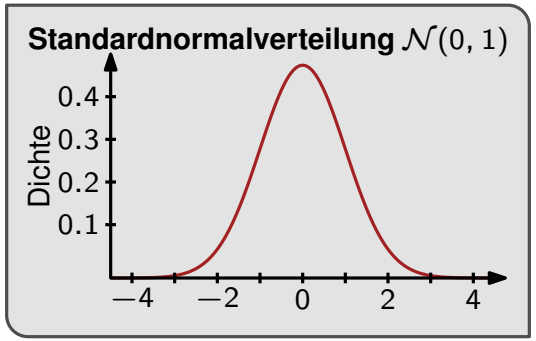


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung
- Unsere Pixel sind 256-seitige Würfel
  (wenn auch vermutlich nicht gleich verteilt)
- Unsere Hashunfktionen
  - die erste bildet Fotos auf ~~große~~ Zahlen ab
    (viele → die gleichen)
  - die zweite bildet nun die ~~großen~~ Zahlen auf (die gleichen) Buckets ab
- Viele Fotos landen in den gleichen Buckets ... Laufzeitreduktion funktioniert nicht

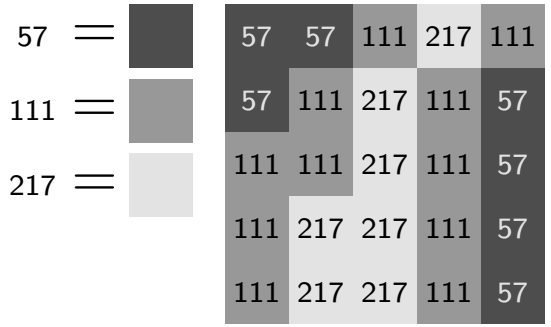


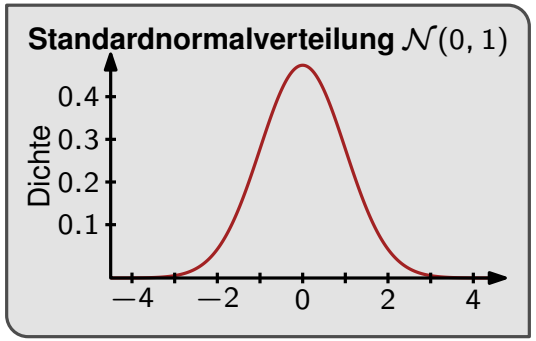


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung
- Unsere Pixel sind 256-seitige Würfel
  (wenn auch vermutlich nicht gleich verteilt)
- Unsere Hashunfktionen
  - die erste bildet Fotos auf ~~große~~ Zahlen ab
    (viele: die gleichen)
  - die zweite bildet nun die ~~großen~~ Zahlen auf Buckets ab
    (die gleichen)
- Viele Fotos landen in den gleichen Buckets ... Laufzeitreduktion funktioniert nicht





Hashfunktionen besser nicht selbst entwerfen!

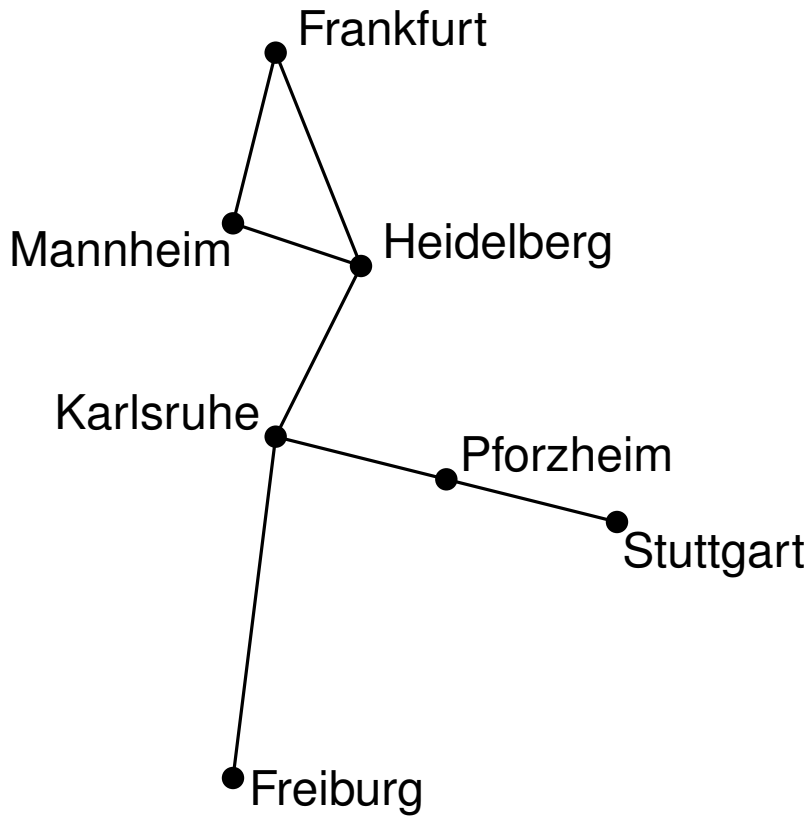
Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Graphen

- flexibles Tool zur Modellierung

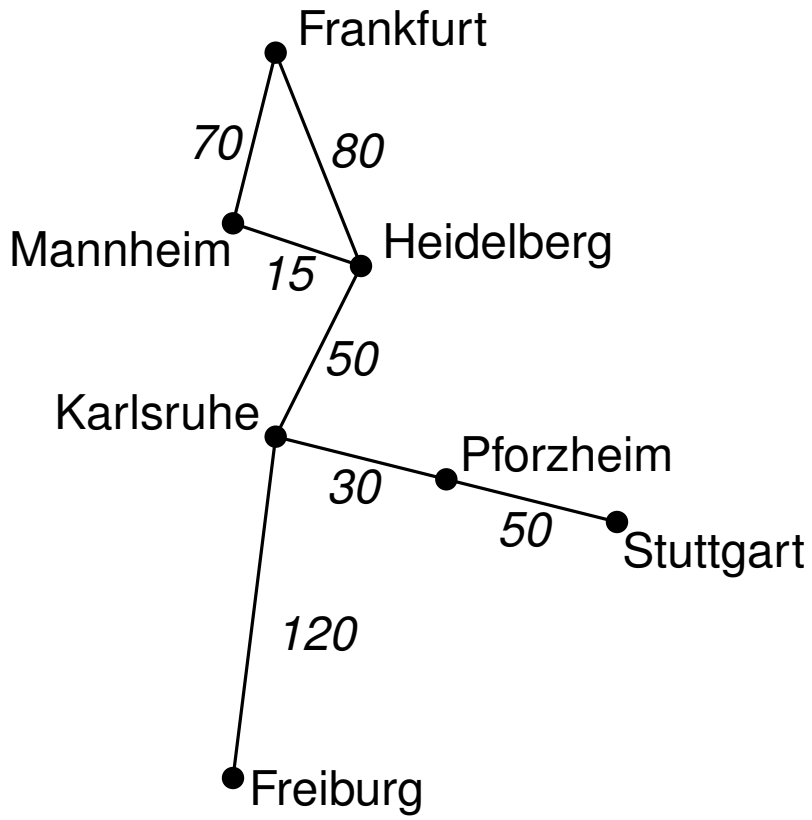
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke



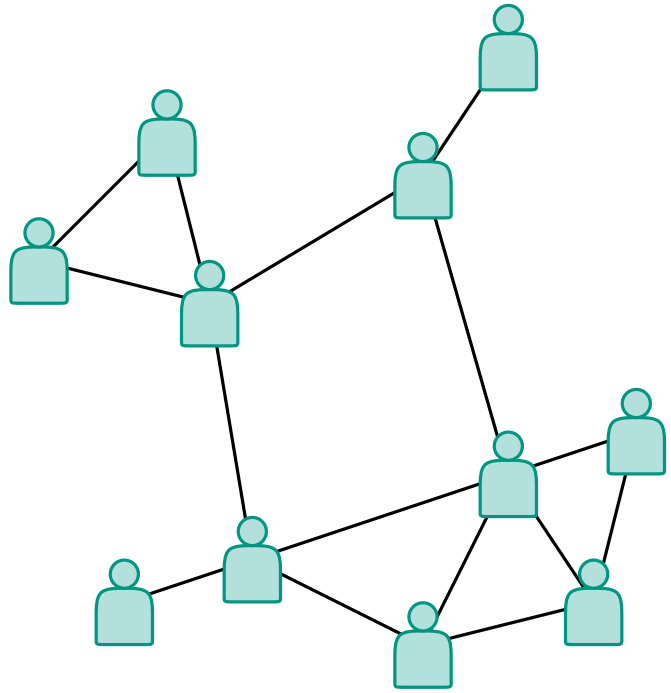
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke



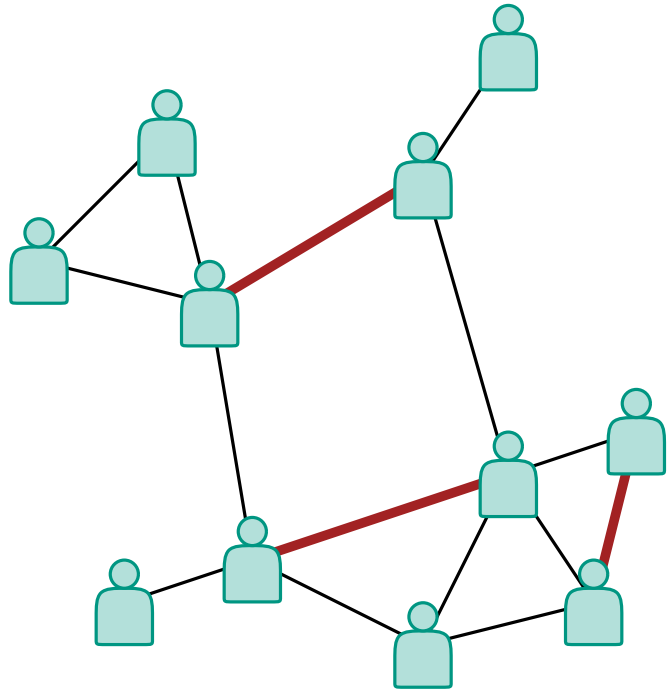
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke



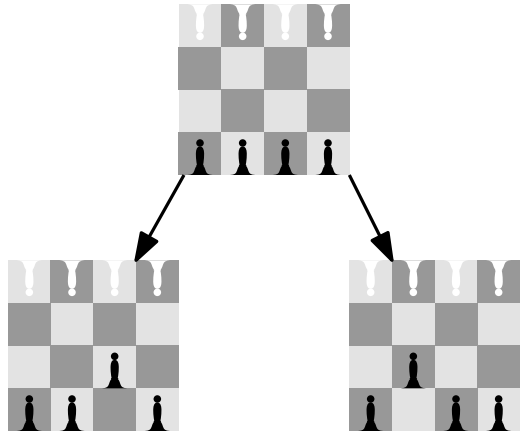
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke



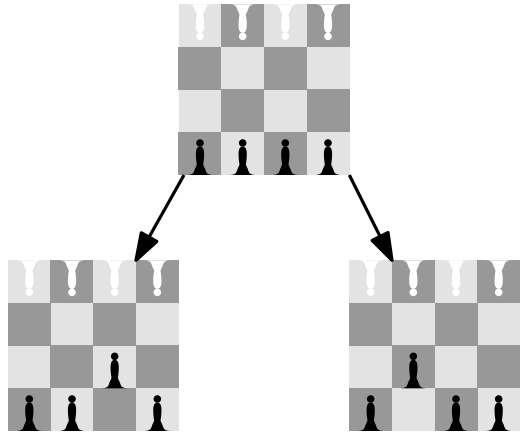
Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke
  - sonstiges



Graphen

- flexibles Tool zur Modellierung
  - Transportnetzwerke
  - soziale Netzwerke
  - sonstiges
- viele praktische Fragestellungen sind algorithmische Probleme auf Graphen



Grundlagen und Notation

Grundlagen und Notation

Graph

$G = (V, E)$

Grundlagen und Notation

Graph

$G = (V, E)$

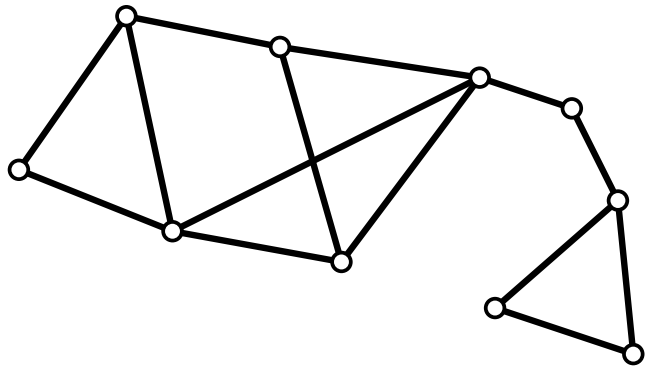
- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

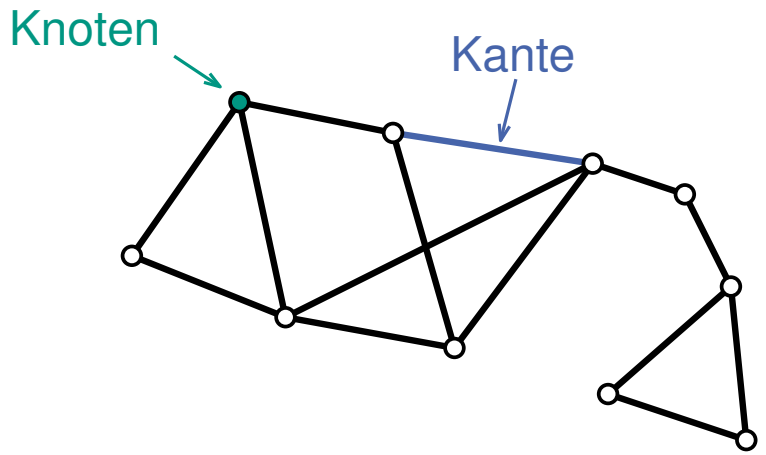


Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

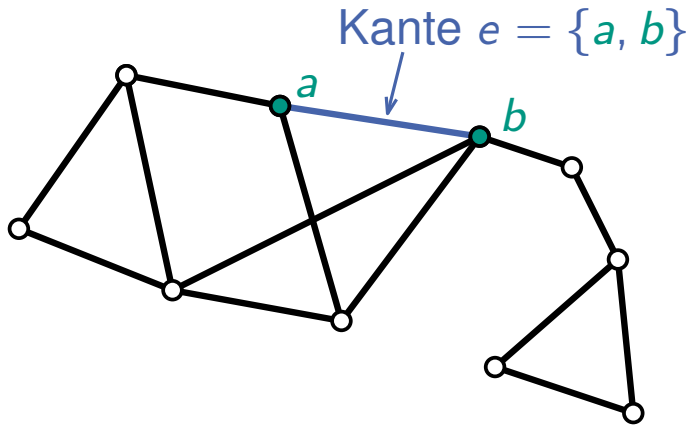


Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

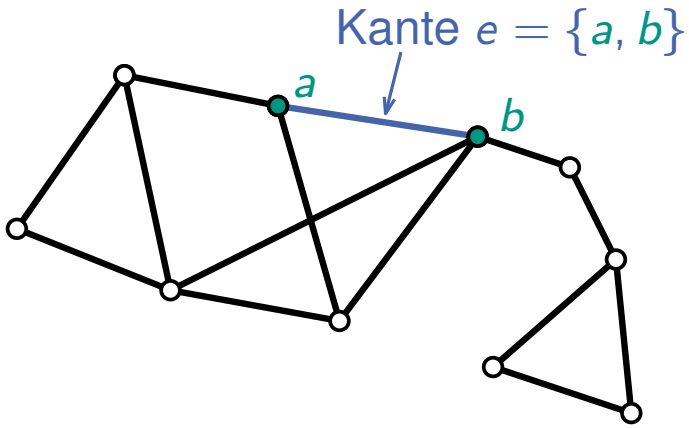


Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



Relationen

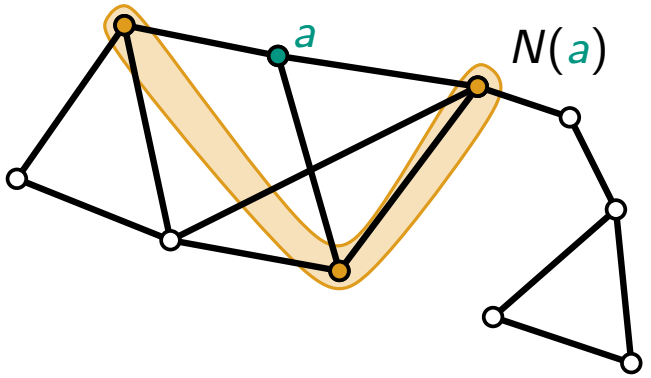
- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



Relationen

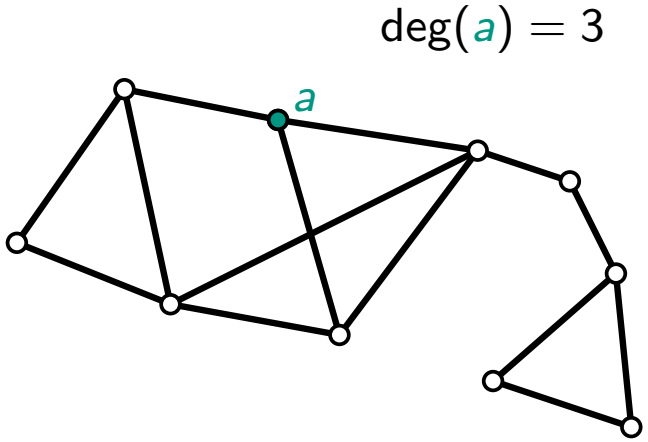
- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$



Relationen

- a adjazent zu b
- e inzident zu a, b
- $b \in N(a)$

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Grundlagen und Notation

Graph

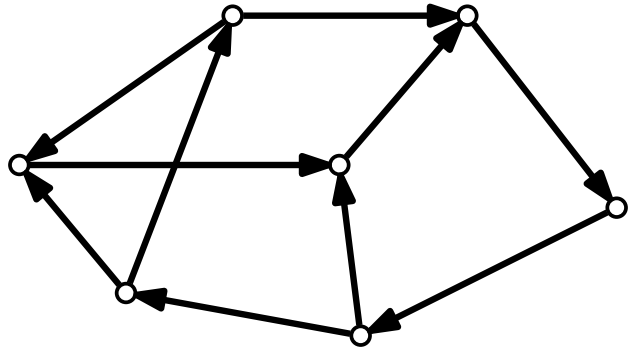
$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$



Grundlagen und Notation

Graph

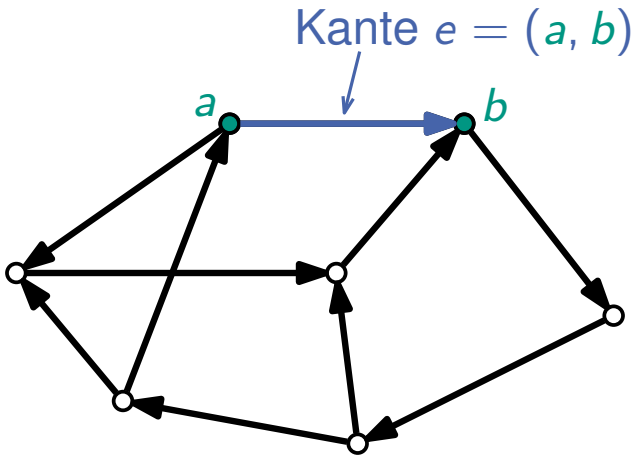
$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$



Grundlagen und Notation

Graph

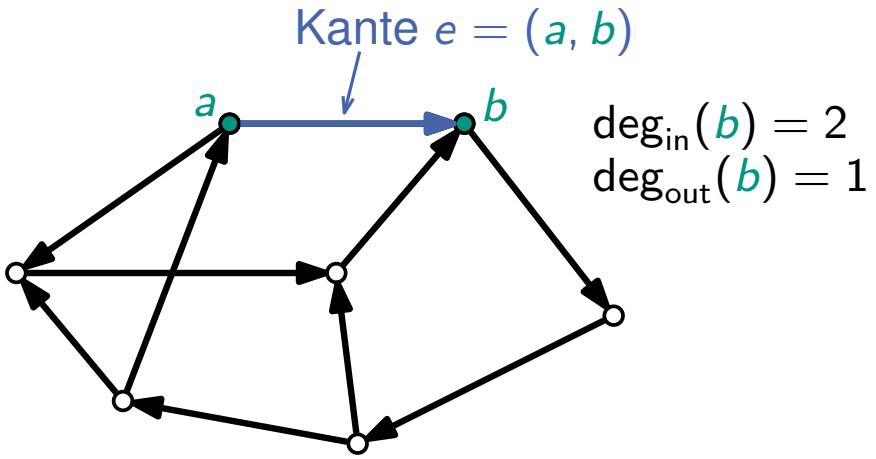
$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$



Grundlagen und Notation

Graph

$G = (V, E)$

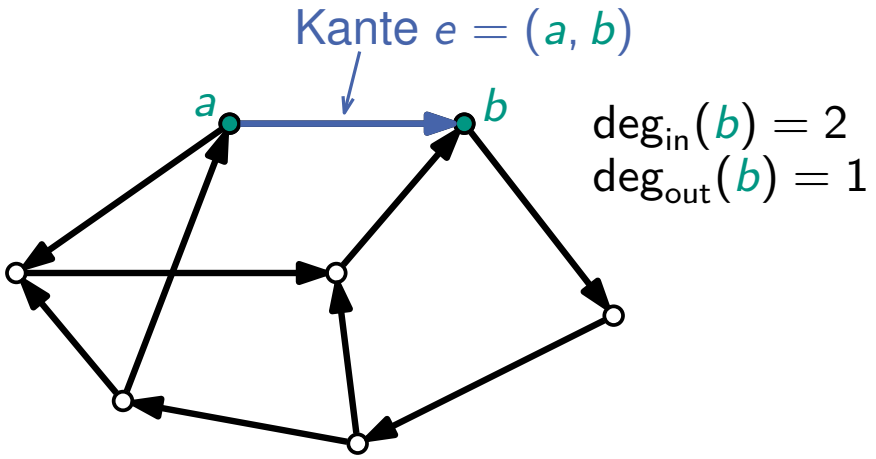
- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Sonstiges



Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

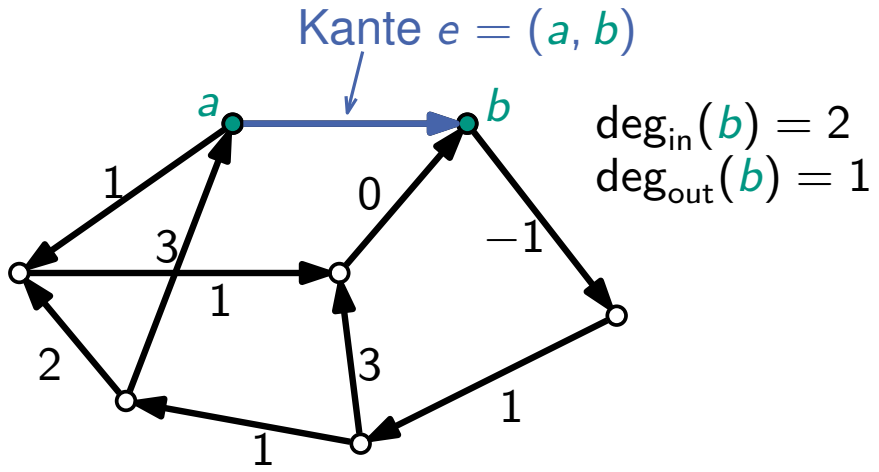
Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$



Grundlagen und Notation

Graph

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $\binom{V}{2}$

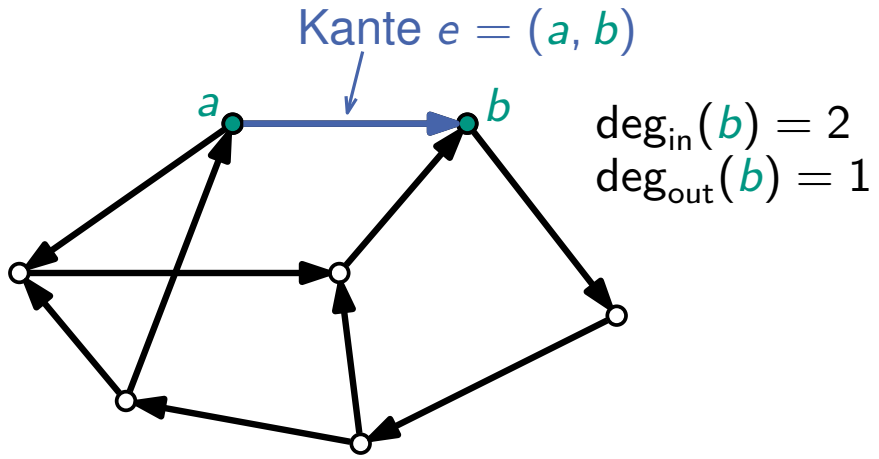
Graph (gerichtet)

$G = (V, E)$

- V: endliche Menge
- E: Teilmenge von $V \times V$

Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$



Grundlagen und Notation

Graph

$G = (V, E)$

- E: Teilmenge von $\binom{V}{2}$
- V: endliche Menge

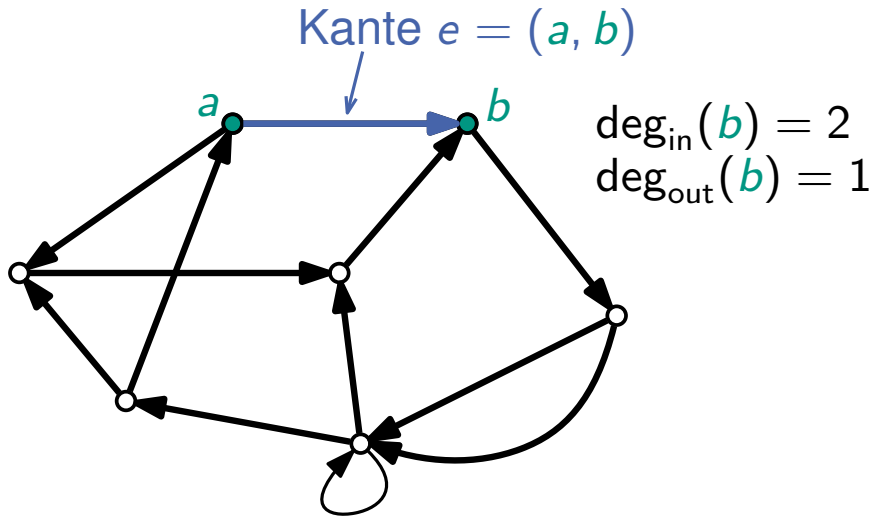
Graph (gerichtet)

$G = (V, E)$

- E: Teilmenge von $V \times V$
- V: endliche Menge

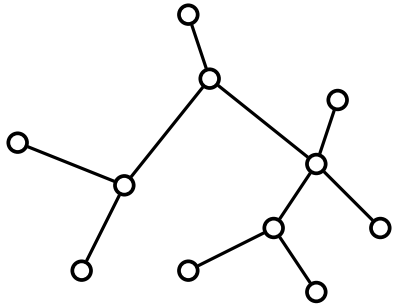
Sonstiges

- Gewichte: $G = (V, E, w)$ mit $w : E \mapsto \mathbb{R}$
- Multigraphen



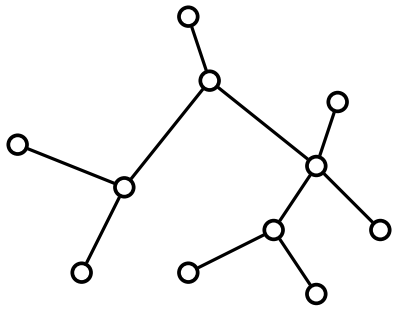
Besondere Graphen

Besondere Graphen



Bäume

Besondere Graphen

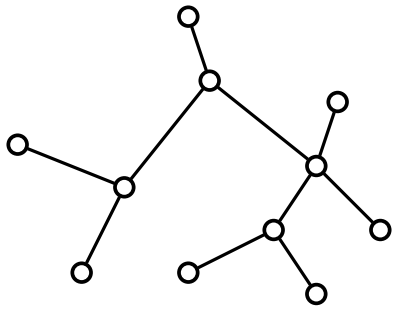


Bäume

Charakterisierung:

- kreisfrei
- zusammenhängend
- $m = n - 1$

Besondere Graphen



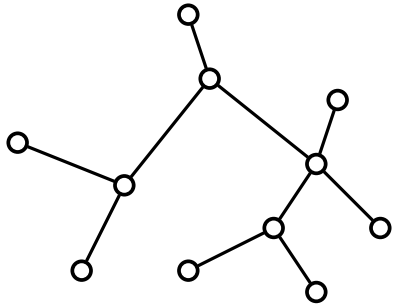
Bäume

Charakterisierung:

- kreisfrei
- zusammenhängend
- $m = n - 1$

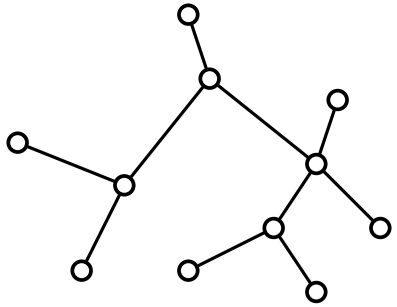
} Zwei Eigenschaften implizieren dritte

Besondere Graphen

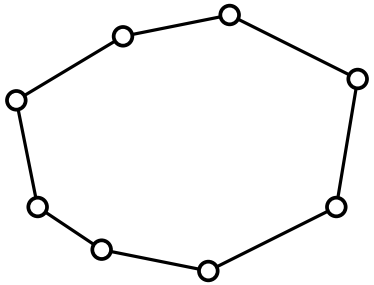


Bäume

Besondere Graphen

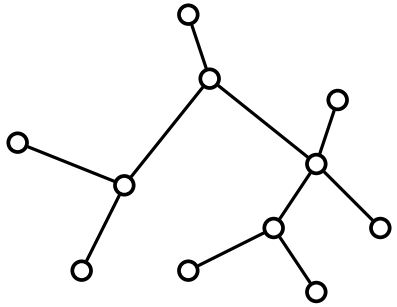


Bäume

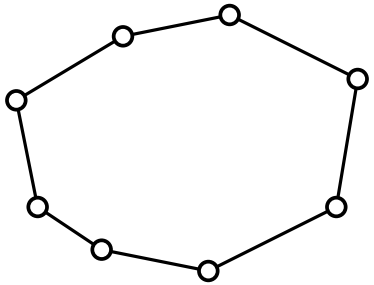


Kreise C_n

Besondere Graphen



Bäume

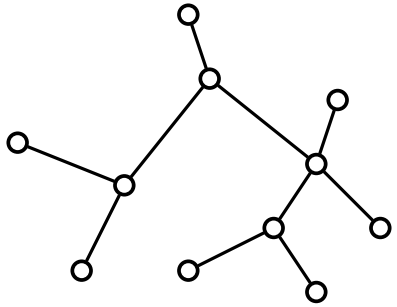


Kreise C_n

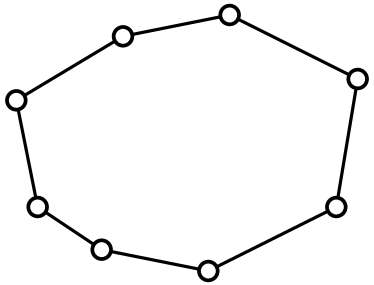
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Besondere Graphen



Bäume



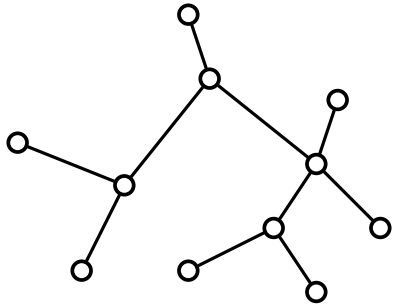
Kreise C_n

Charakterisierung:

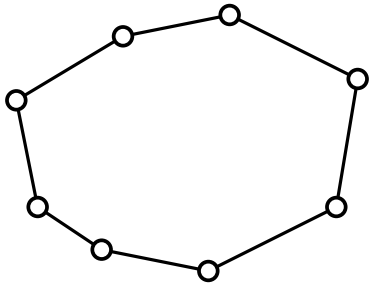
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

Besondere Graphen



Bäume



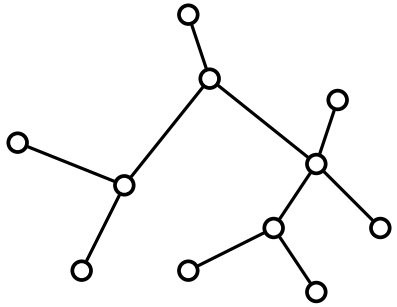
Kreise C_n

Charakterisierung:

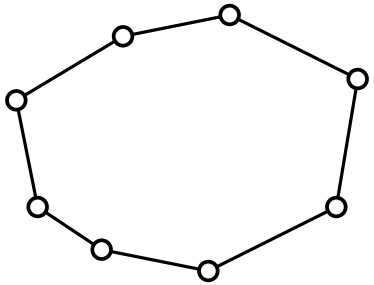
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

Besondere Graphen



Bäume



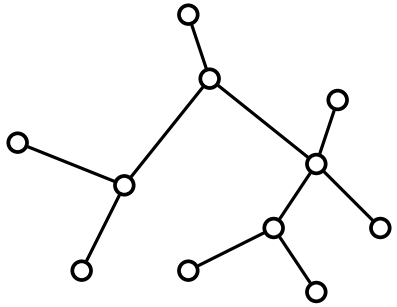
Kreise C_n

Charakterisierung:

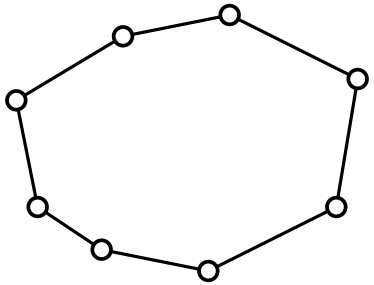
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

Besondere Graphen



Bäume

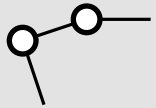


Kreise C_n

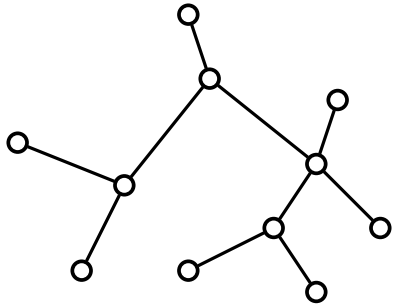
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

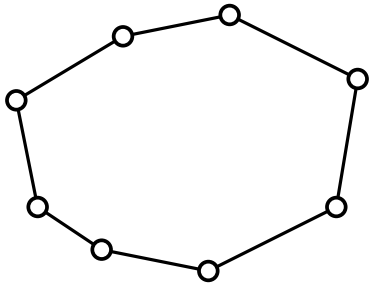
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

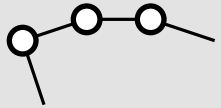


Kreise C_n

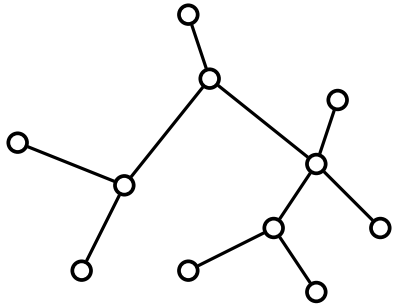
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

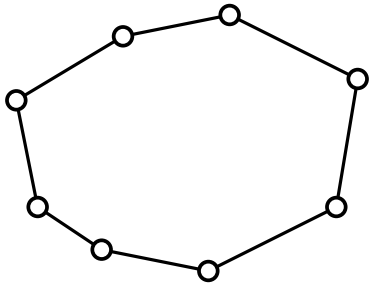
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

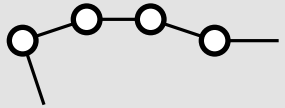


Kreise C_n

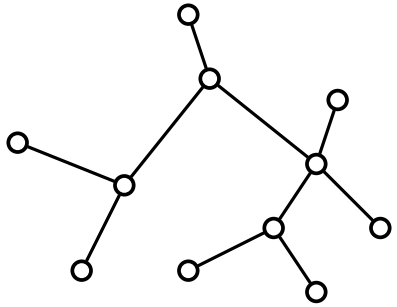
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

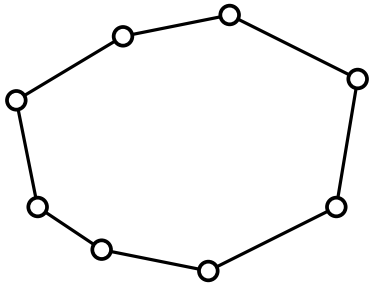
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

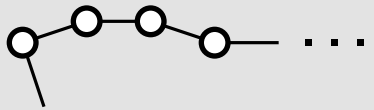


Kreise C_n

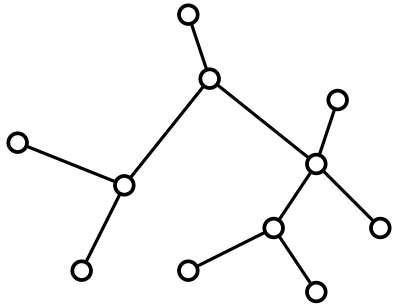
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

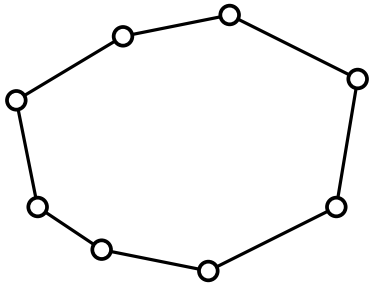
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

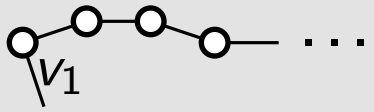


Kreise C_n

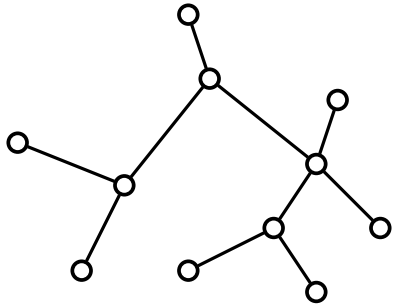
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

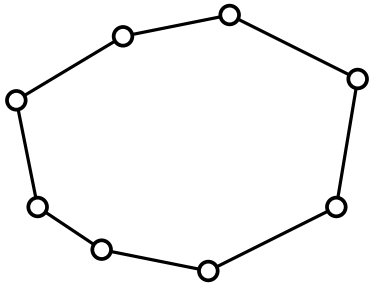
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

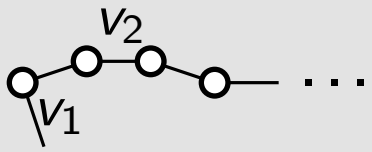


Kreise C_n

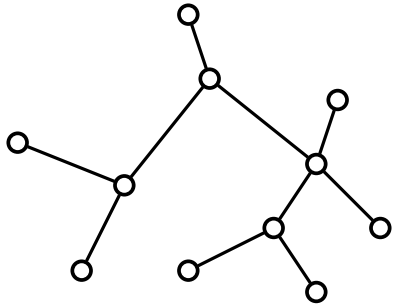
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

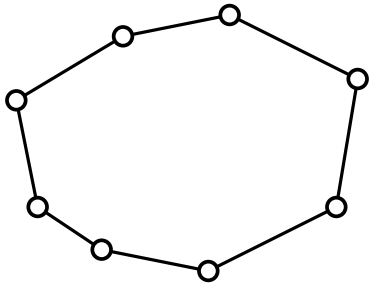
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

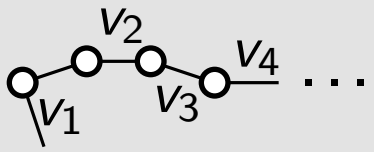


Kreise C_n

Charakterisierung:

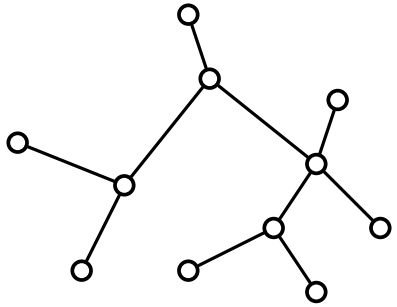
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

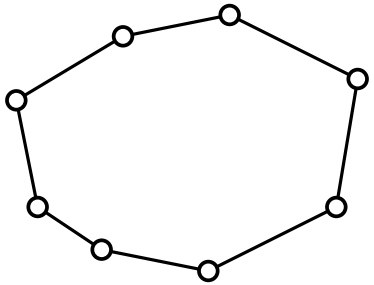


...

Besondere Graphen



Bäume

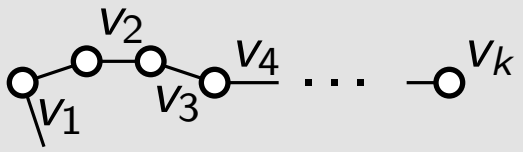


Kreise C_n

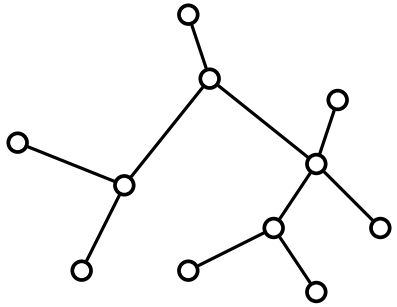
Charakterisierung:

- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

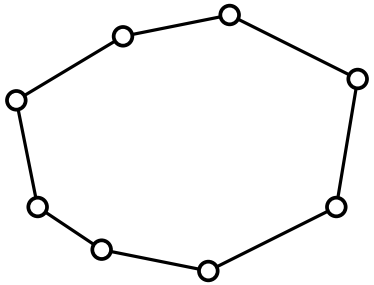
Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis



Besondere Graphen



Bäume

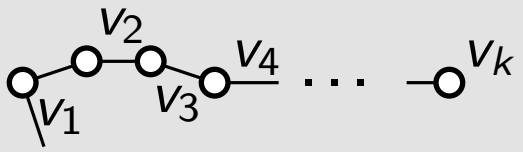


Kreise C_n

Charakterisierung:

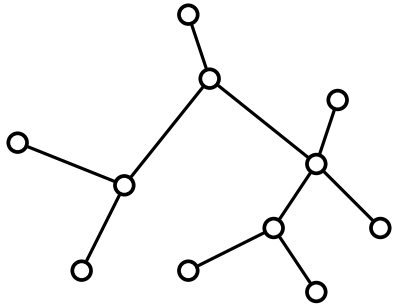
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

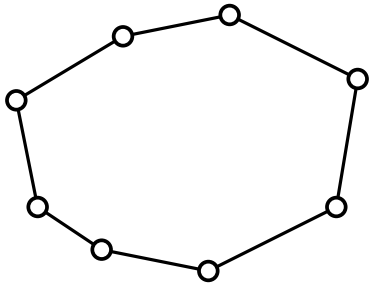


- für $2 \leq i < k - 1$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

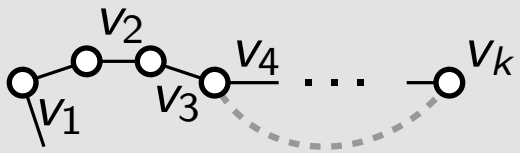


Kreise C_n

Charakterisierung:

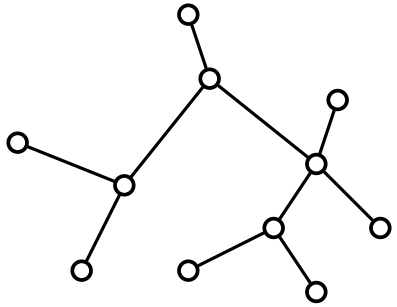
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

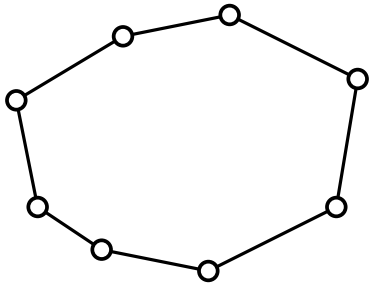


- für $2 \leq i < k - 1$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

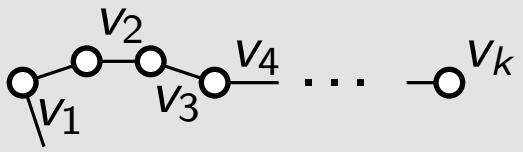


Kreise C_n

Charakterisierung:

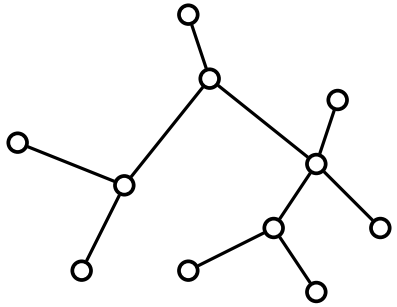
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

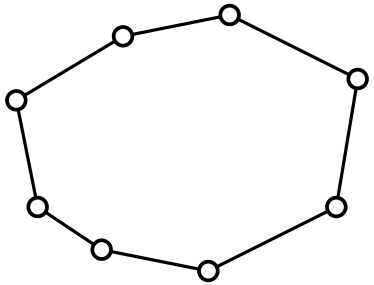


- für $2 \leq i < k - 1$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

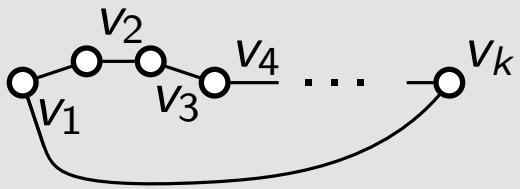


Kreise C_n

Charakterisierung:

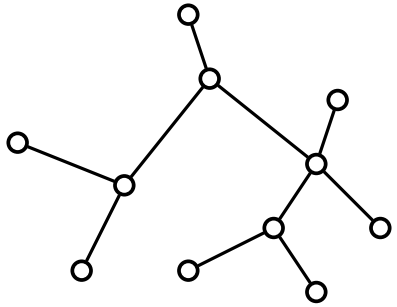
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

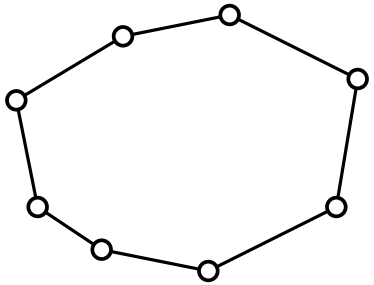


- für $2 \leq i < k - 1$: $\{v_i, v_k\} \notin E$

Besondere Graphen



Bäume

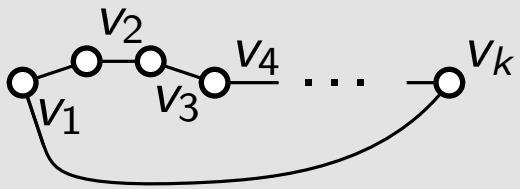


Kreise C_n

Charakterisierung:

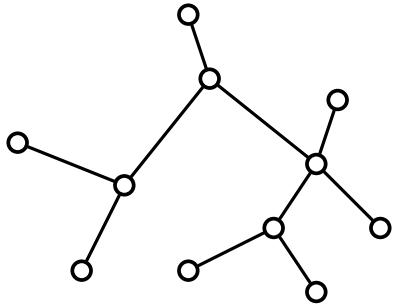
- zusammenhängend
- $\deg(v) = 2$ f.a. $v \in V$

Argumentation: $\forall v \in V : \deg(v) = 2 \Rightarrow G$ ist Kreis

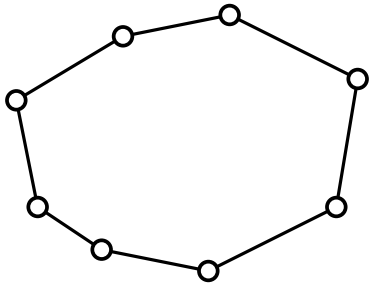


- für $2 \leq i < k-1$: $\{v_i, v_k\} \notin E$
- $\{v_k, v_1\} \in E$ falls $k = n$

Besondere Graphen

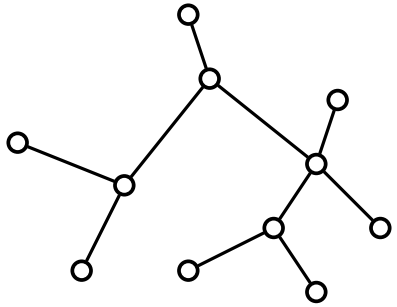


Bäume

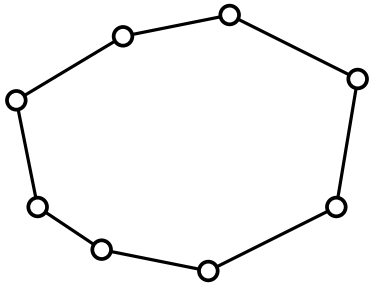


Kreise

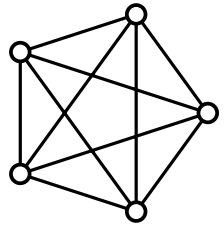
Besondere Graphen



Bäume

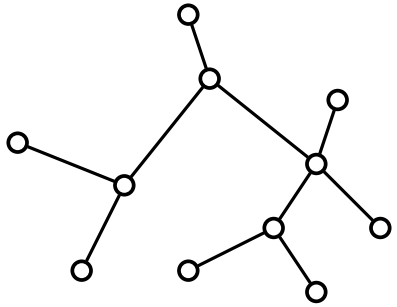


Kreise

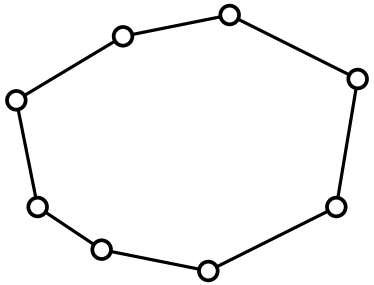


vollst. Graph K_n

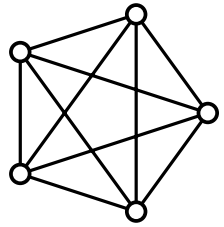
Besondere Graphen



Bäume



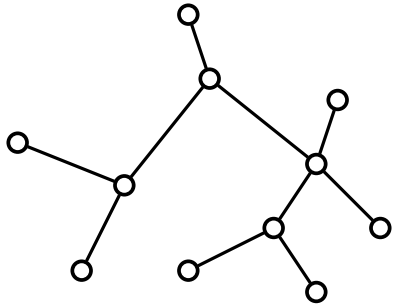
Kreise



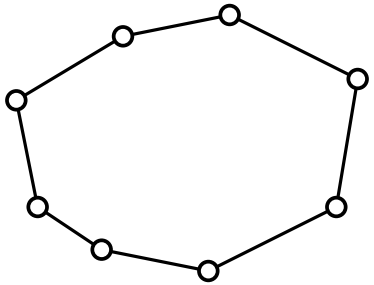
vollst. Graph K_n

$$m = \binom{n}{2} = \frac{n(n-1)}{2}$$

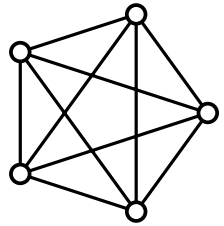
Besondere Graphen



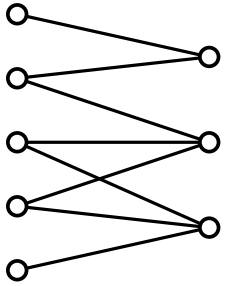
Bäume



Kreise

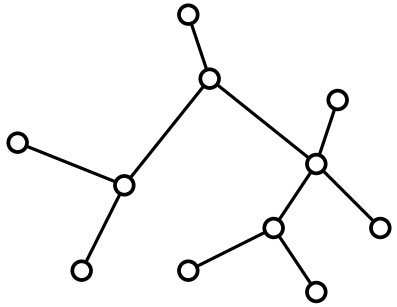


vollst. Graph K_n

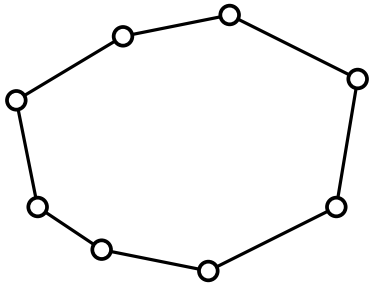


bipartiter Graph

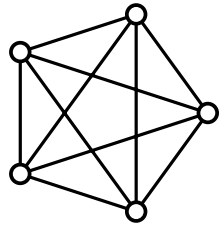
Besondere Graphen



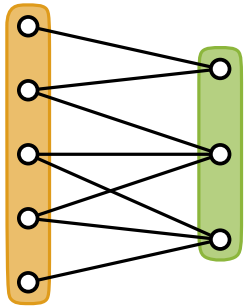
Bäume



Kreise



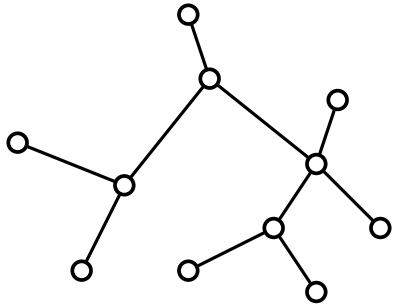
vollst. Graph K_n



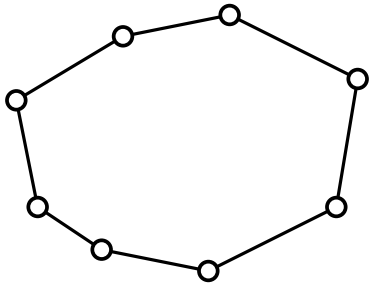
bipartiter Graph

$G = (A \cup B, E)$

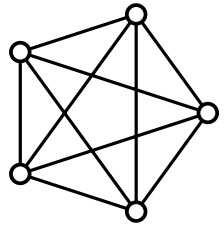
Besondere Graphen



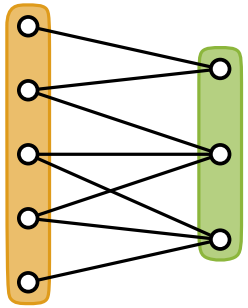
Bäume



Kreise



vollst. Graph K_n

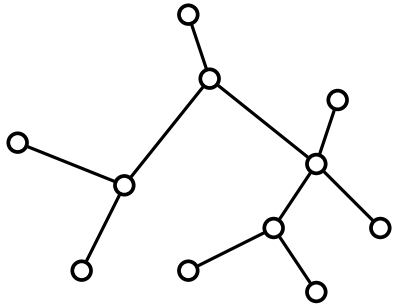


bipartiter Graph

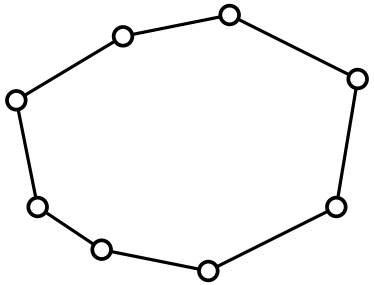
$G = (A \cup B, E)$

Charakterisierung:
G enthält nur Kreise *gerader* Länge in G

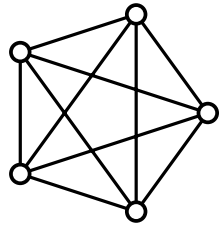
Besondere Graphen



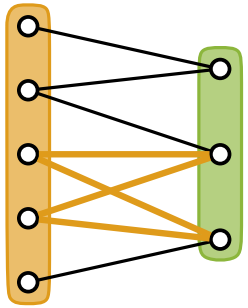
Bäume



Kreise



vollst. Graph K_n

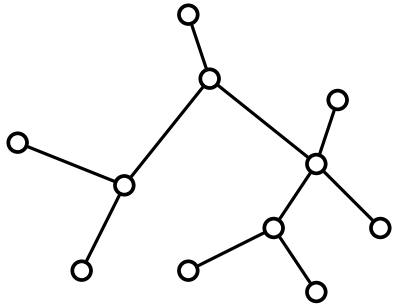


bipartiter Graph

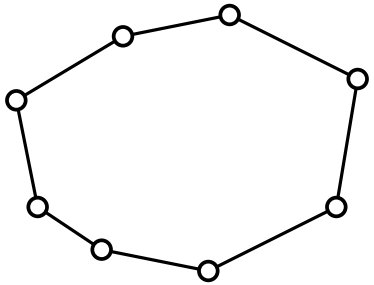
$G = (A \cup B, E)$

Charakterisierung:
G enthält nur Kreise *gerader* Länge in G

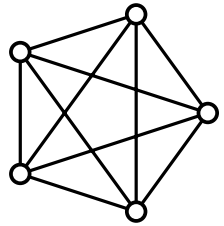
Besondere Graphen



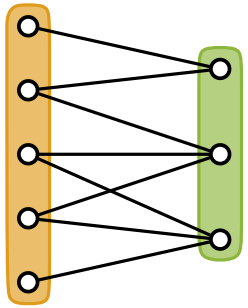
Bäume



Kreise



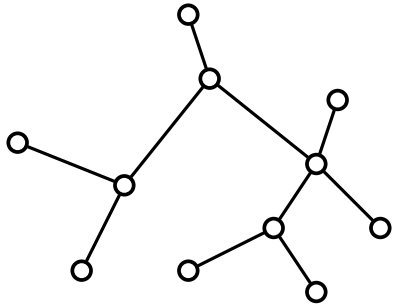
vollst. Graph K_n



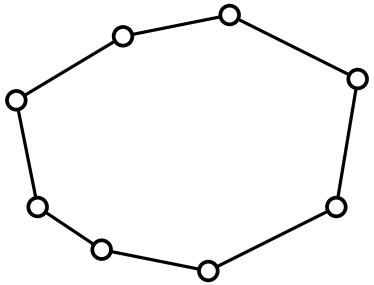
bipartiter Graph

$G = (A \cup B, E)$

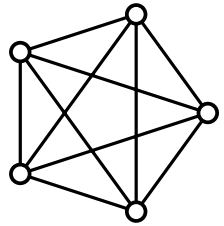
Besondere Graphen



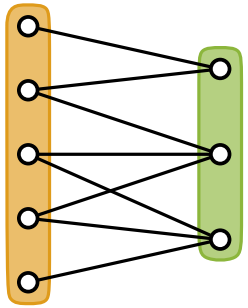
Bäume



Kreise

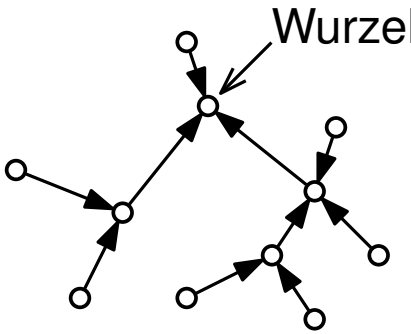


vollst. Graph K_n



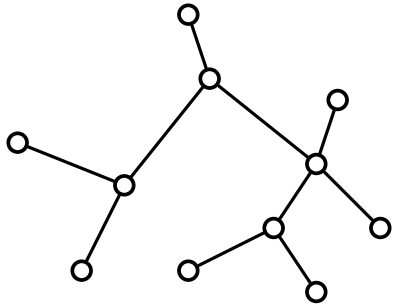
bipartiter Graph

$G = (A \cup B, E)$

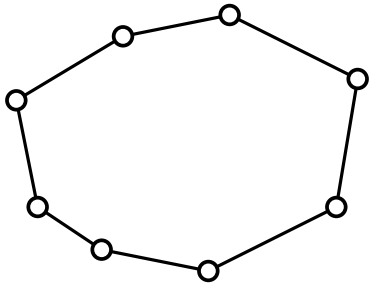


In-tree / zur Wurzel ger. Baum

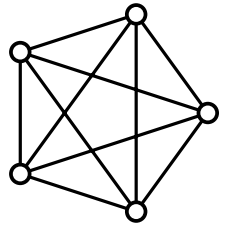
Besondere Graphen



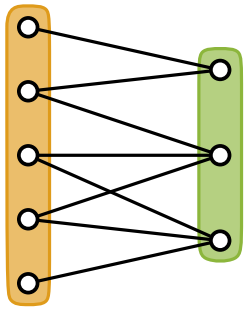
Bäume



Kreise

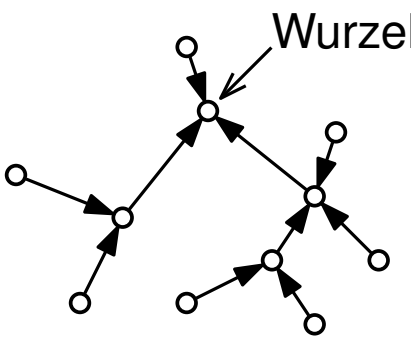


vollst. Graph K_n

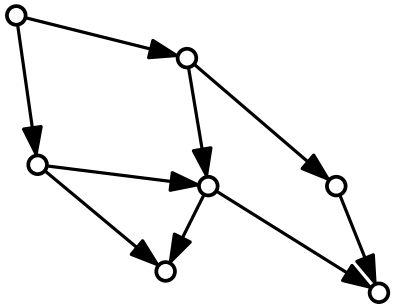


bipartiter Graph

$G = (A \cup B, E)$

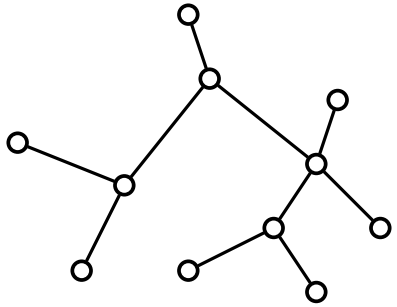


In-tree / zur Wurzel ger. Baum

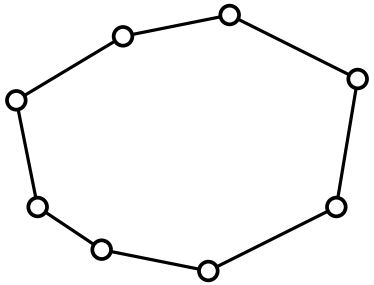


directed acyclic graph (DAG)

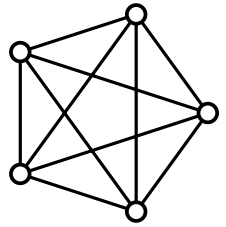
Besondere Graphen



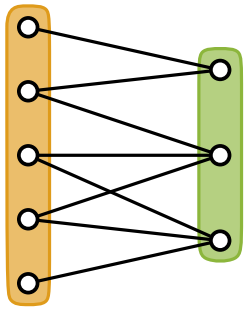
Bäume



Kreise

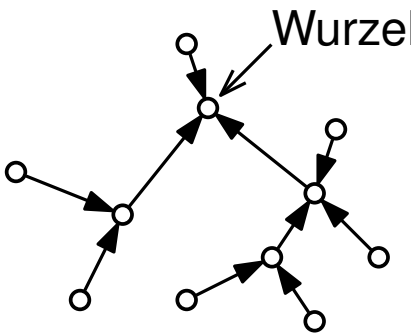


vollst. Graph K_n

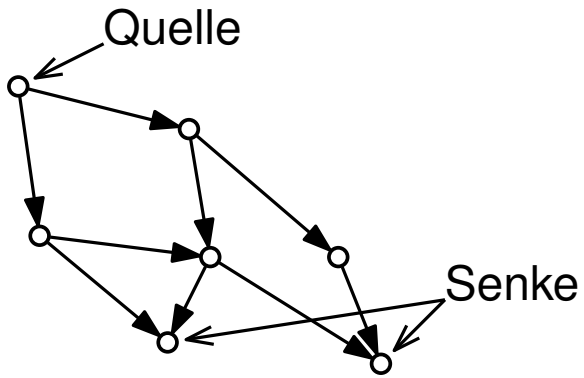


bipartiter Graph

$G = (A \cup B, E)$



In-tree / zur Wurzel ger. Baum



directed acyclic graph (DAG)

Problem: Dependencies

Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL:	https://aur.archlinux.org/ipe.git (read-only, click to copy)
Package Base:	ipe
Description:	The extensible drawing editor
Upstream URL:	http://ipe.otfried.org/
Licenses:	GPL
Conflicts:	ipe
Submitter:	foxcub
Maintainer:	foxcub
Last Packager:	foxcub
Votes:	77
Popularity:	0.033711
First Submitted:	2007-01-02 01:58 (UTC)
Last Updated:	2021-06-16 18:14 (UTC)

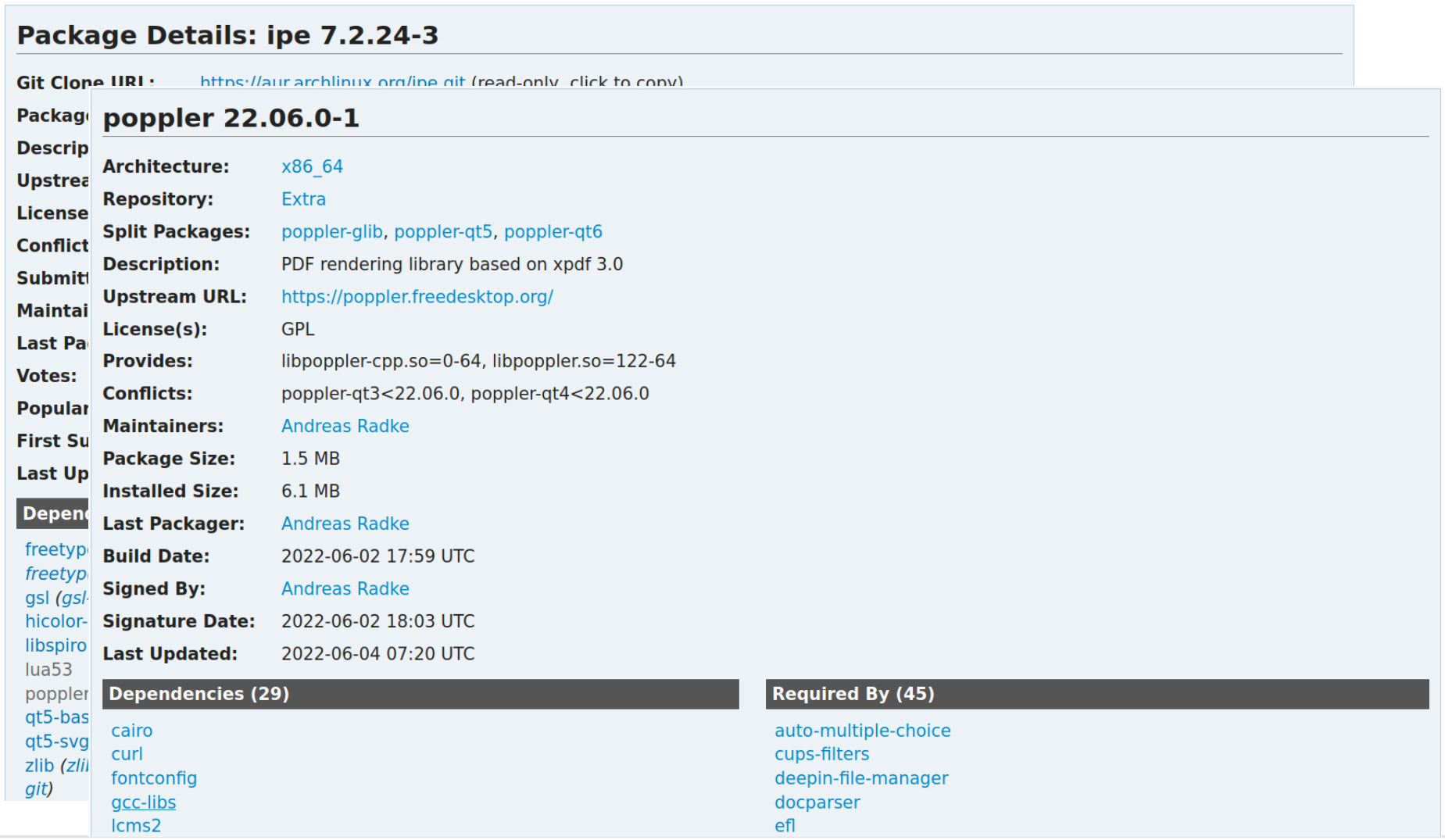
Dependencies (9)

freetype2 (*freetype2-minimal-git*, *freetype2-ttmetrics*, *freetype2-v35*, *freetype2-git*, *freetype2-ultimate5*, *freetype2-infinality-remix*)
gsl (*gsl-git*)
hicolor-icon-theme (*hicolor-icon-theme-git*)
libspiro
lua53
poppler (*poppler-minimal*, *poppler-lcdfilter*, *poppler-lcd*, *poppler-git*)
qt5-base (*qt5-base-git*, *qt5-base-headless*)
qt5-svg (*qt5-svg-git*)
zlib (*zlib-static*, *zlib-git*, *zlib-asm*, *minizip-asm*, *zlib-ng*, *zlib-ng-compat-git*)

Required by (4)

cgal-ipelets
cgal-ipelets (*make*)
ipe-tools-git
ipe2tikz-git

Problem: Dependencies



Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Packag
Descrip
Upstrea
License
Conflict
Submitt
Maintai
Last Pa
Votes:
Popular
First Su
Last Up

Depen

freetyp
freetyp
gsl (*gsl*
hicolor-
libspiro
lua53
poppler
qt5-bas
qt5-svg
zlib (*zli*
git)

poppler 22.06.0-1

Architecture:	x86_64
Repository:	Extra
Split Packages:	poppler-glib, poppler-qt5, poppler-qt6
Description:	PDF rendering library based on xpdf 3.0
Upstream URL:	https://poppler.freedesktop.org/
License(s):	GPL
Provides:	libpoppler-cpp.so=0-64, libpoppler.so=122-64
Conflicts:	poppler-qt3<22.06.0, poppler-qt4<22.06.0
Maintainers:	Andreas Radke
Package Size:	1.5 MB
Installed Size:	6.1 MB
Last Packager:	Andreas Radke
Build Date:	2022-06-02 17:59 UTC
Signed By:	Andreas Radke
Signature Date:	2022-06-02 18:03 UTC
Last Updated:	2022-06-04 07:20 UTC

Dependencies (29)

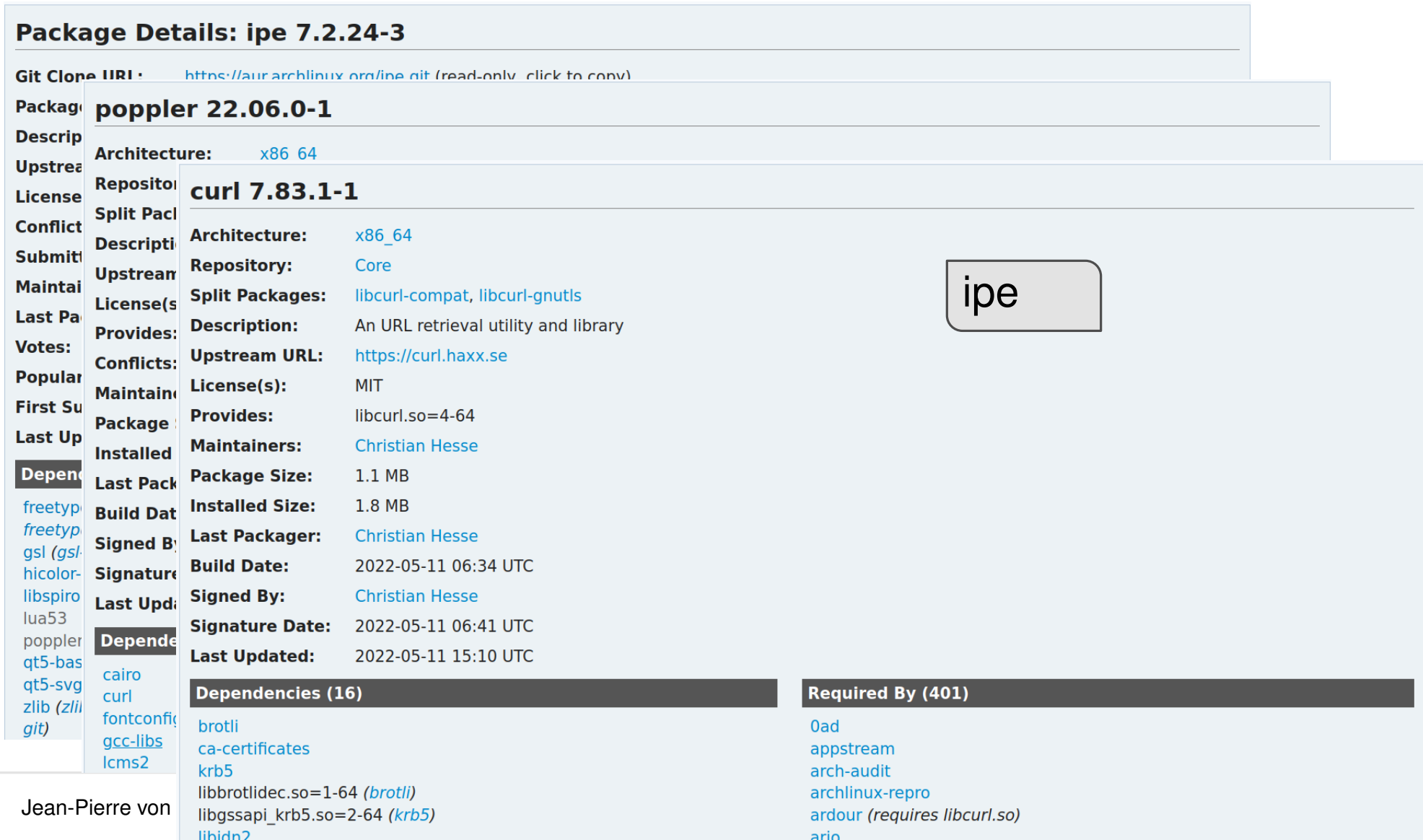
cairo
curl
fontconfig
gcc-libs
lcms2

Required By (45)

auto-multiple-choice
cups-filters
deepin-file-manager
docparser
efl

Problem: Dependencies

Problem: Dependencies



Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Package

Descrip

Upstrea

License

Conflict

Submitt

Maintai

Last Pa

Votes:

Popular

First Su

Last Up

Depen

freetyp

freetyp

gsl *(gsl*

hicolor-

libspiro

lua53

poppler

qt5-bas

qt5-svg

zlib *(zli*

git)

poppler 22.06.0-1

Architecture: x86_64

Repositor

Split Pacl

Descripti

Upstream

License(s

Provides:

Conflicts:

Maintain

Package

Installed

Last Pack

Build Dat

Signed By

Signature

Last Upd

Depende

cairo

curl

fontconfig

gcc-libs

lcms2

curl 7.83.1-1

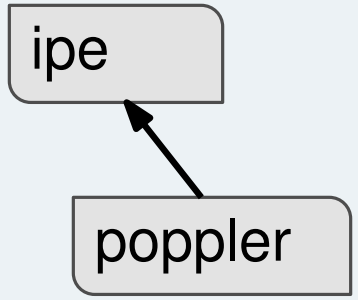
Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian Hesse
Build Date:	2022-05-11 06:34 UTC
Signed By:	Christian Hesse
Signature Date:	2022-05-11 06:41 UTC
Last Updated:	2022-05-11 15:10 UTC

Dependencies (16)

brotli
ca-certificates
krb5
libbrotlidec.so=1-64 *(brotli)*
libgssapi_krb5.so=2-64 *(krb5)*
libidn2

Required By (401)

0ad
appstream
arch-audit
archlinux-repro
ardour *(requires libcurl.so)*
ario



Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Packag
Descrip
Upstrea
License
Conflict
Submitt
Maintai
Last Pa
Votes:
Popular
First Su
Last Up

Depen

freetyp
freetyp
gsl (*gsl*
hicolor-
libspiro
lua53
poppler
qt5-bas
qt5-svg
zlib (*zli*
git)

poppler 22.06.0-1

Architecture: x86_64

Repositor
Split Pacl
Descripti
Upstream
License(s
Provides:
Conflicts:
Maintain
Package
Installed
Last Pack
Build Dat
Signed By
Signature
Last Upd

Depende

cairo
curl
fontconfig
gcc-libs
lcms2

curl 7.83.1-1

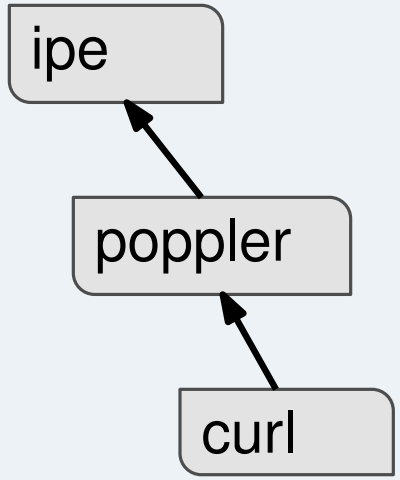
Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian Hesse
Build Date:	2022-05-11 06:34 UTC
Signed By:	Christian Hesse
Signature Date:	2022-05-11 06:41 UTC
Last Updated:	2022-05-11 15:10 UTC

Dependencies (16)

brotli
ca-certificates
krb5
libbrotlidec.so=1-64 (*brotli*)
libgssapi_krb5.so=2-64 (*krb5*)
libidn2

Required By (401)

0ad
appstream
arch-audit
archlinux-repro
ardour (*requires libcurl.so*)
arjo



Problem: Dependencies

Package Details: ipe 7.2.24-3

Git Clone URL: https://aur.archlinux.org/ipe.git (read-only, click to copy)

Package

Descrip

Upstrea

License

Conflict

Submitt

Maintai

Last Pa

Votes:

Popular

First Su

Last Up

Depen

freetyp
freetyp
gsl *(gsl*
hicolor-
libspiro
lua53
poppler
qt5-bas
qt5-svg
zlib *(zli*
git)

poppler 22.06.0-1

Architecture: x86_64

Repositor

Split Pacl

Descripti

Upstream

License(s

Provides:

Conflicts:

Maintain

Package

Installed

Last Pack

Build Dat

Signed By

Signature

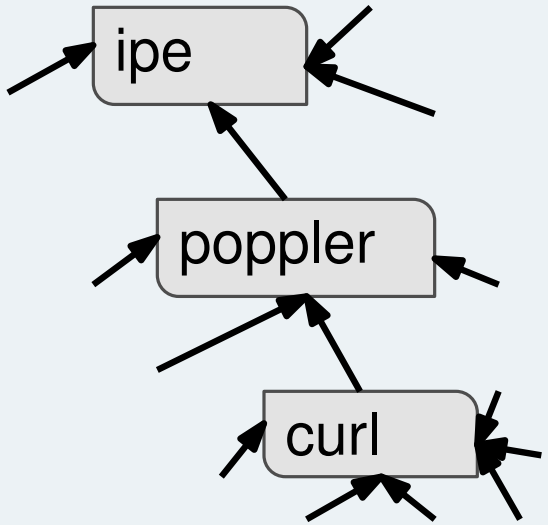
Last Upd

Depende

cairo
curl
fontconfig
gcc-libs
lcms2

curl 7.83.1-1

Architecture:	x86_64
Repository:	Core
Split Packages:	libcurl-compat, libcurl-gnutls
Description:	An URL retrieval utility and library
Upstream URL:	https://curl.haxx.se
License(s):	MIT
Provides:	libcurl.so=4-64
Maintainers:	Christian Hesse
Package Size:	1.1 MB
Installed Size:	1.8 MB
Last Packager:	Christian Hesse
Build Date:	2022-05-11 06:34 UTC
Signed By:	Christian Hesse
Signature Date:	2022-05-11 06:41 UTC
Last Updated:	2022-05-11 15:10 UTC



Dependencies (16)

brotli
ca-certificates
krb5
libbrotlidec.so=1-64 *(brotli)*
libgssapi_krb5.so=2-64 *(krb5)*
libidn2

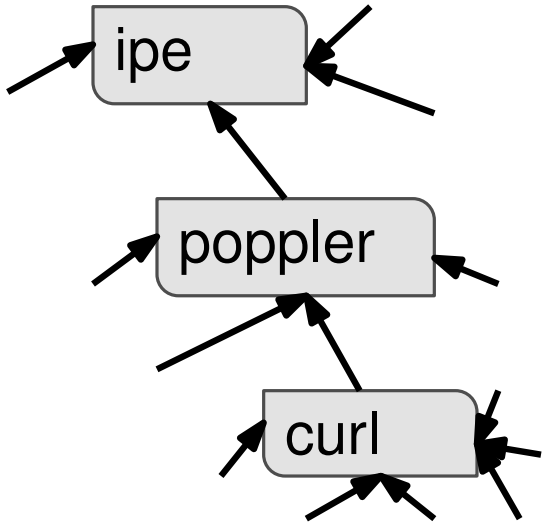
Required By (401)

0ad
appstream
arch-audit
archlinux-repro
ardour *(requires libcurl.so)*
arip

Problem: Dependencies

Modellierung als Graph

- Knoten V: Menge von Paketen
- Knoten E: $(v, w) \in E \Leftrightarrow v$ von w benötigt



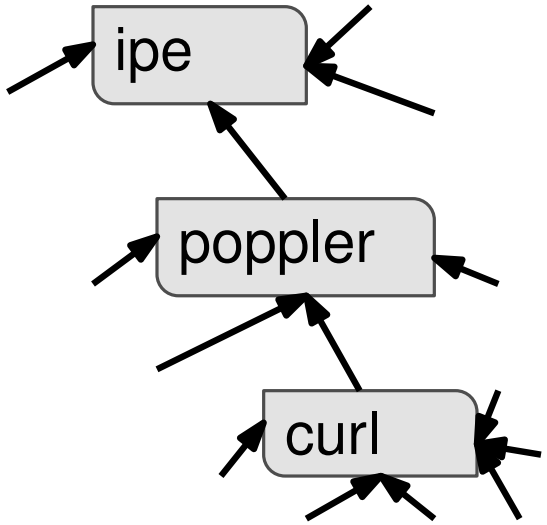
Problem: Dependencies

Modellierung als Graph

- Knoten V: Menge von Paketen
- Knoten E: $(v, w) \in E \Leftrightarrow v$ von w benötigt

Frage

Gibt es zyklische Abhängigkeiten?



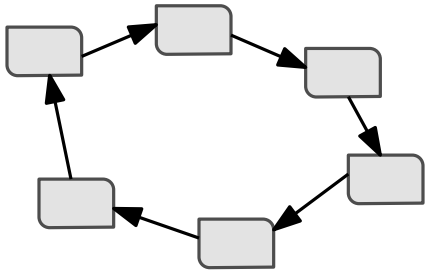
Problem: Dependencies

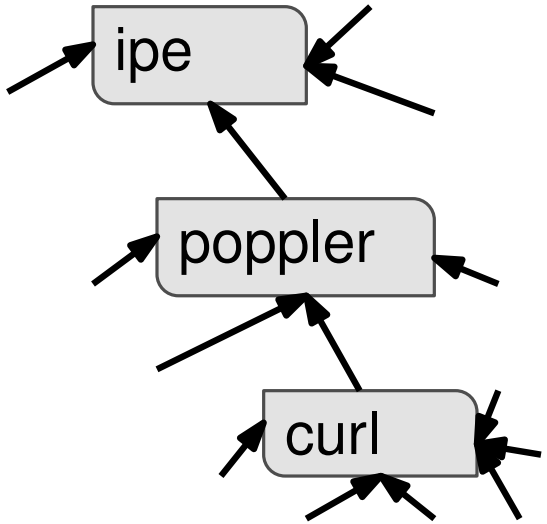
Modellierung als Graph

- Knoten V: Menge von Paketen
- Knoten E: $(v, w) \in E \Leftrightarrow v$ von w benötigt

Frage

Gibt es zyklische Abhängigkeiten?

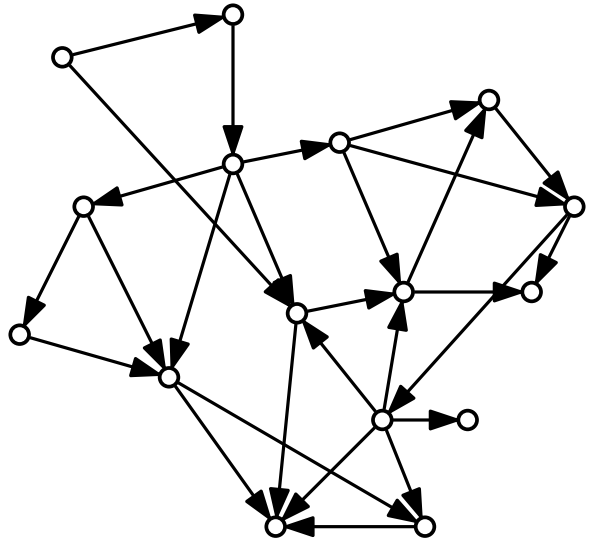




Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

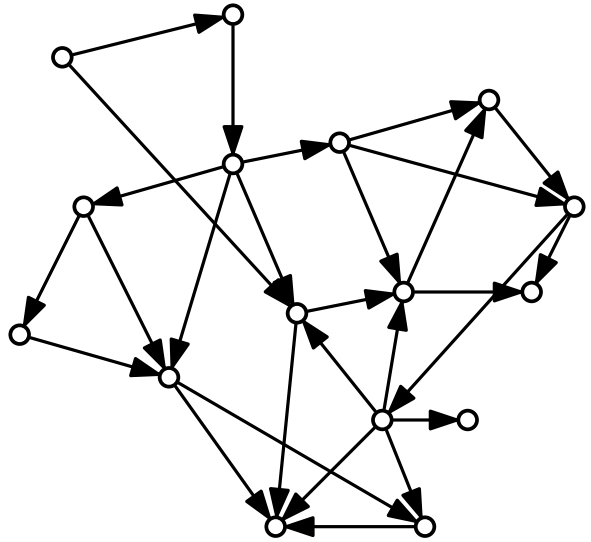


Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze



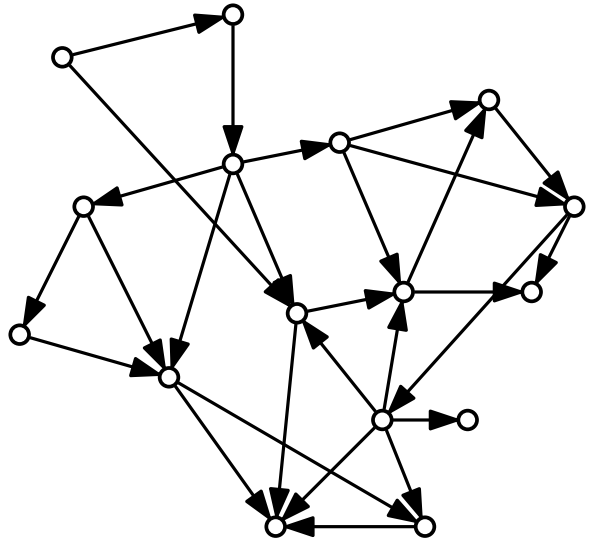
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force



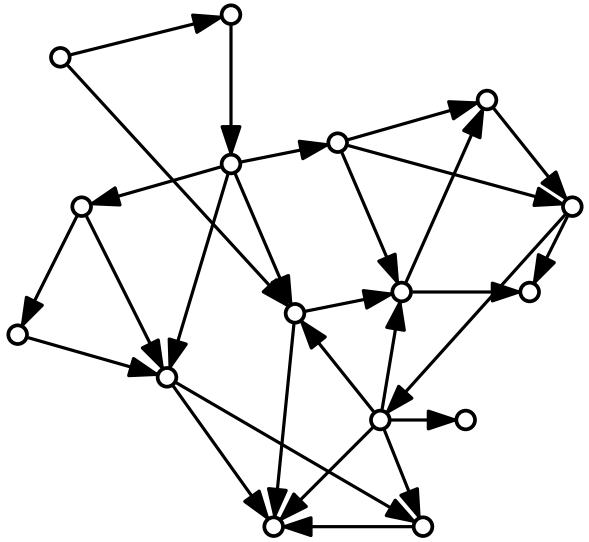
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



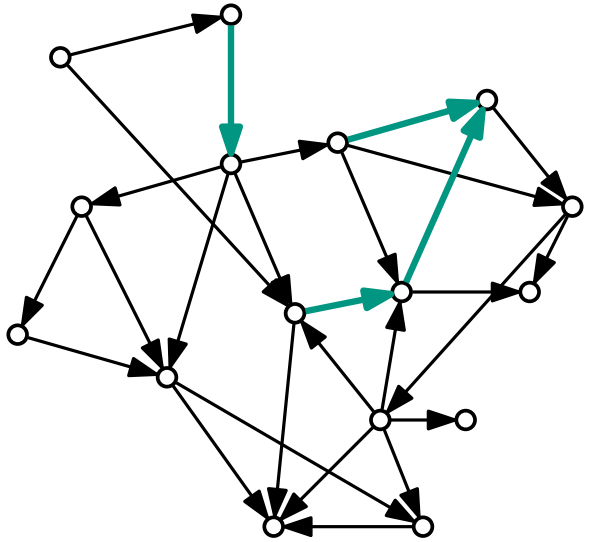
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



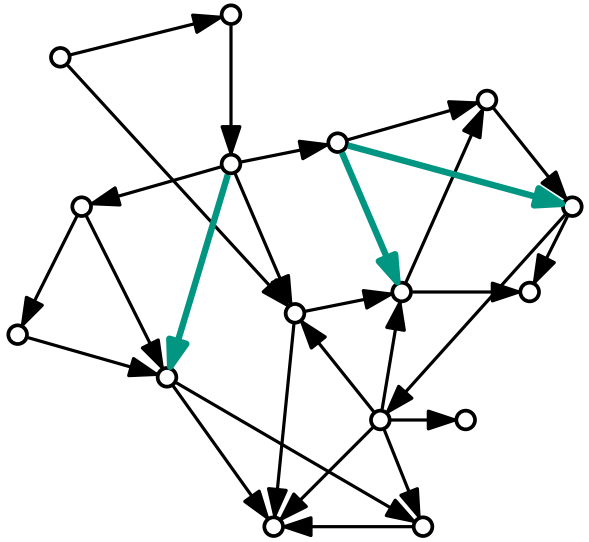
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



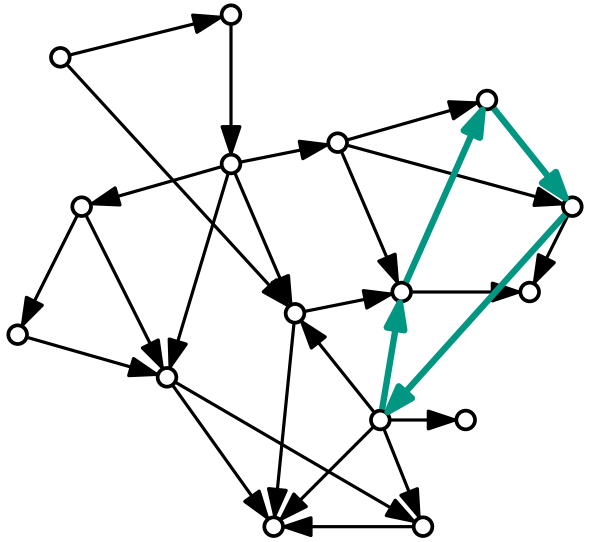
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



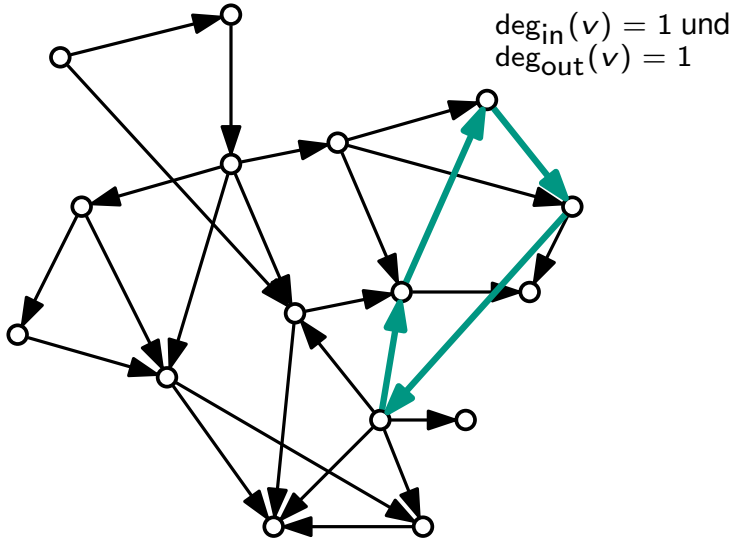
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$
    - prüfe ob E' Kreis ergibt



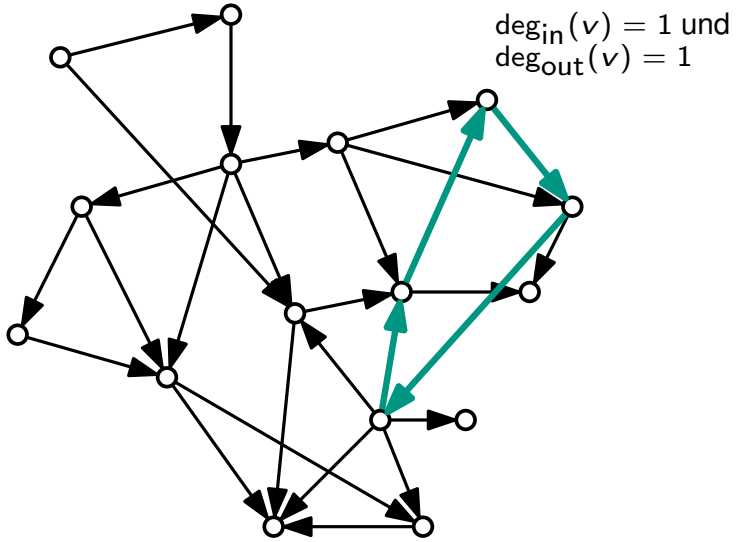
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$ $\quad O(2^m)$
    - prüfe ob E' Kreis ergibt



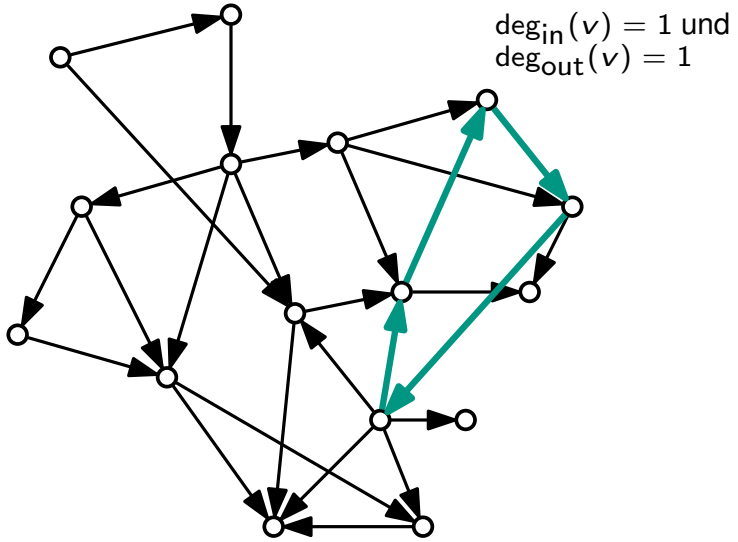
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$ $O(2^m)$
    - prüfe ob E' Kreis ergibt $O(n + m)$



Cyclic Dependency

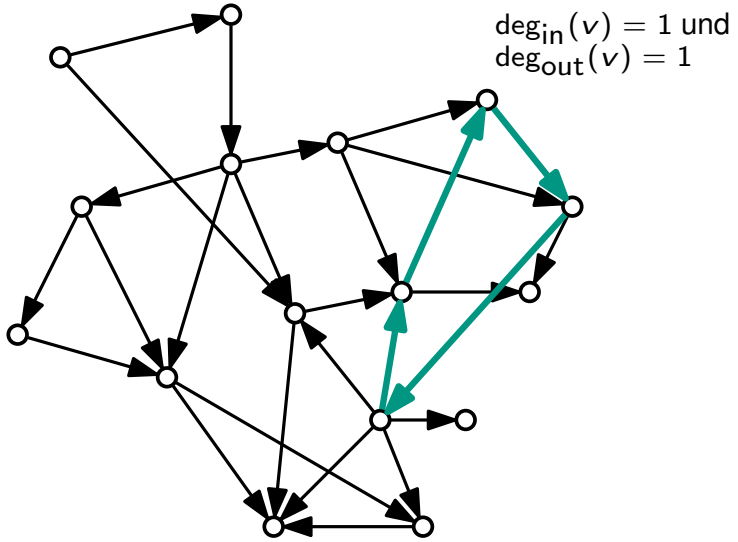
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
  - betrachte jede Teilmenge $E' \subset E$ $O(2^m)$
    - prüfe ob E' Kreis ergibt $O(n + m)$

Gesamt: $O(2^m \cdot (n+m))$



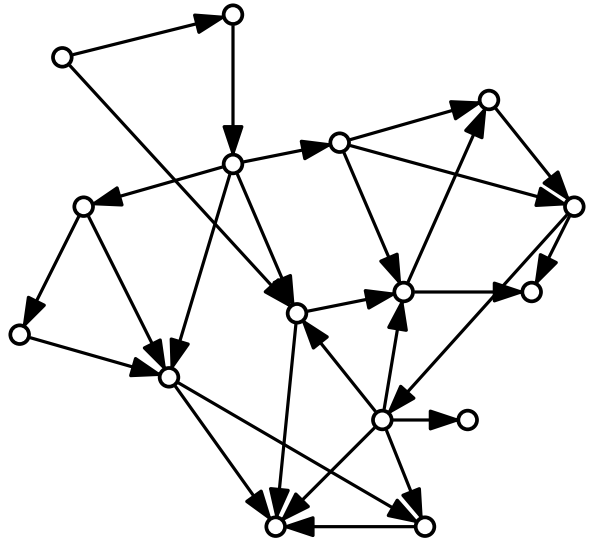
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force



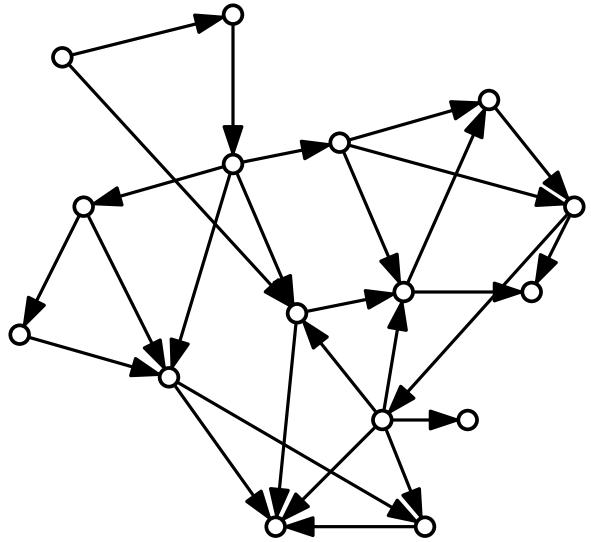
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter



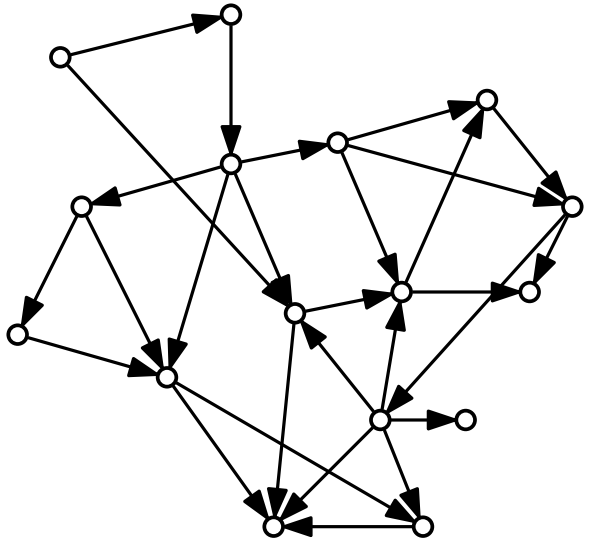
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$



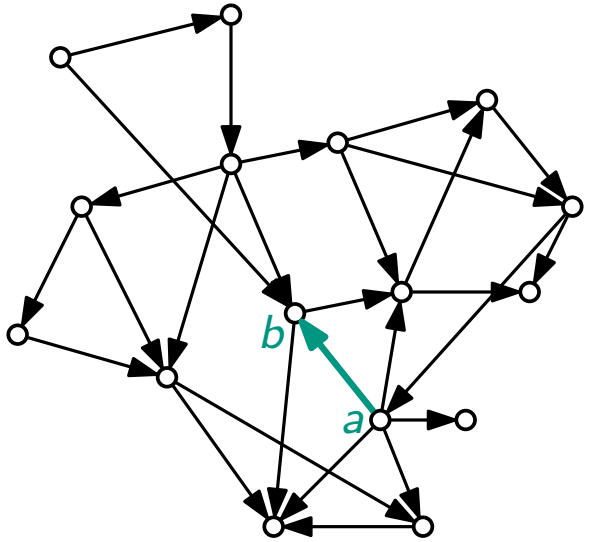
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$



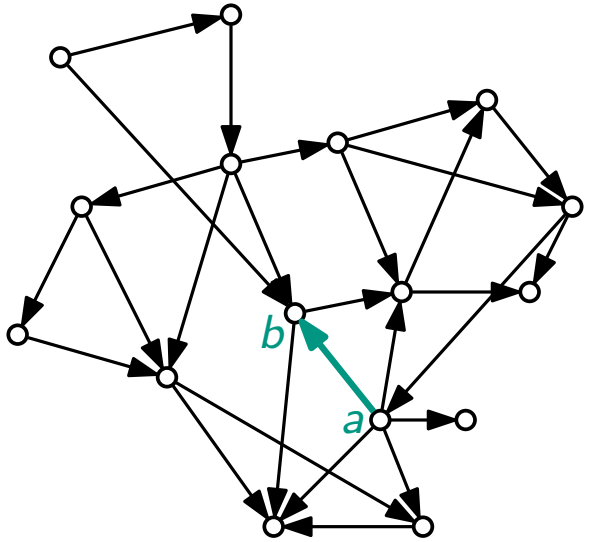
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$
    - suche b-a-Pfad



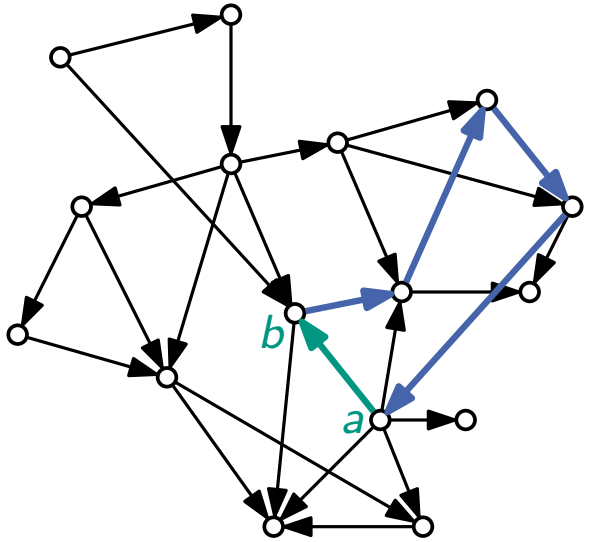
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$
    - suche b-a-Pfad



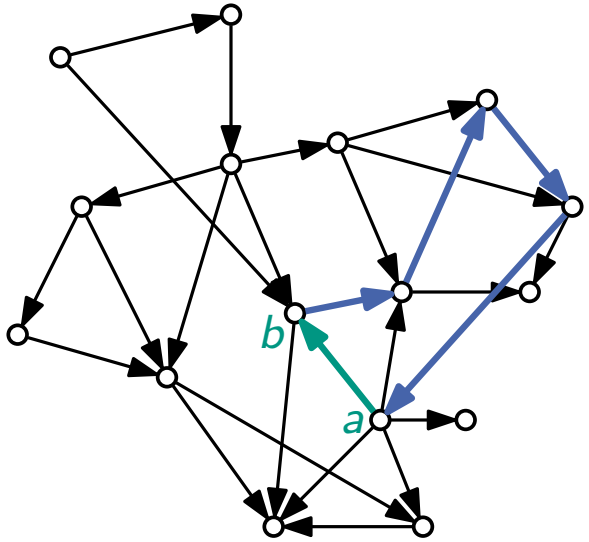
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $\quad O(m)$
    - suche b-a-Pfad



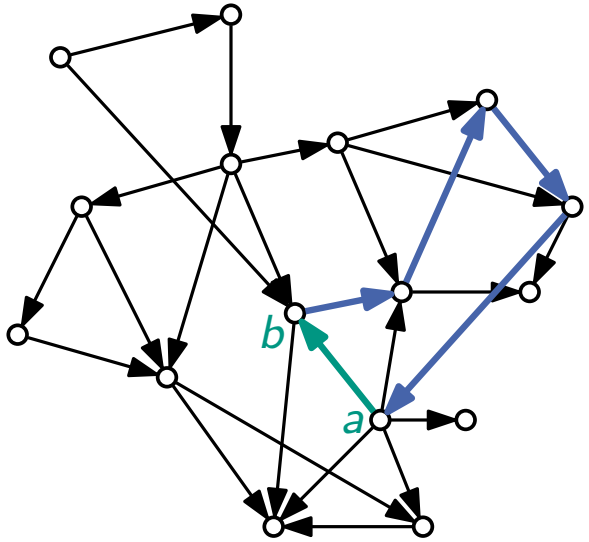
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $O(m)$
    - suche b-a-Pfad $O(n + m)$



Cyclic Dependency

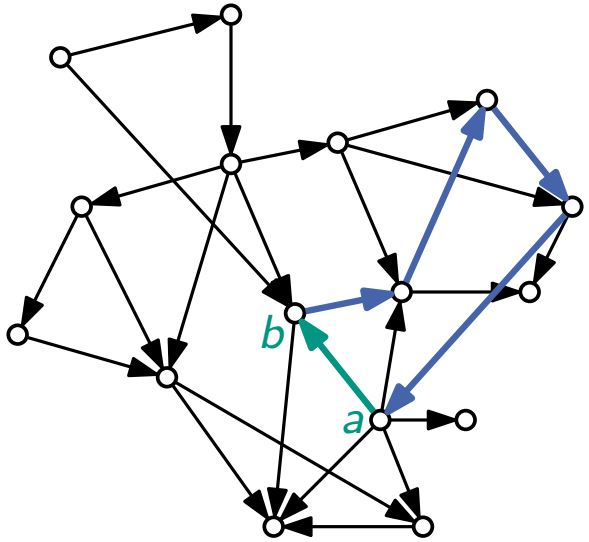
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $O(m)$
    - suche b-a-Pfad $O(n + m)$

Gesamt: $O(m(n+m))$



Cyclic Dependency

Problemstellung

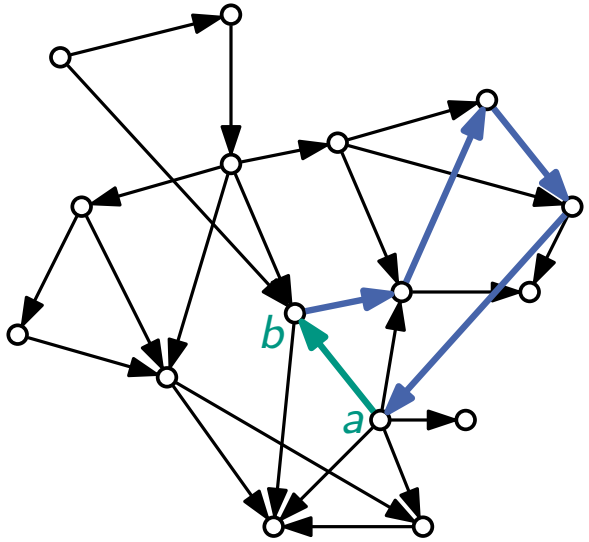
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Lösungsansätze

- brute-force
- etwas geschickter
  - betrachte jede Kante $(a, b) \in E$ $O(m)$
    - suche b-a-Pfad $O(n + m)$

Gesamt: $O(m(n+m))$

Frage: Ist Linearzeit möglich?

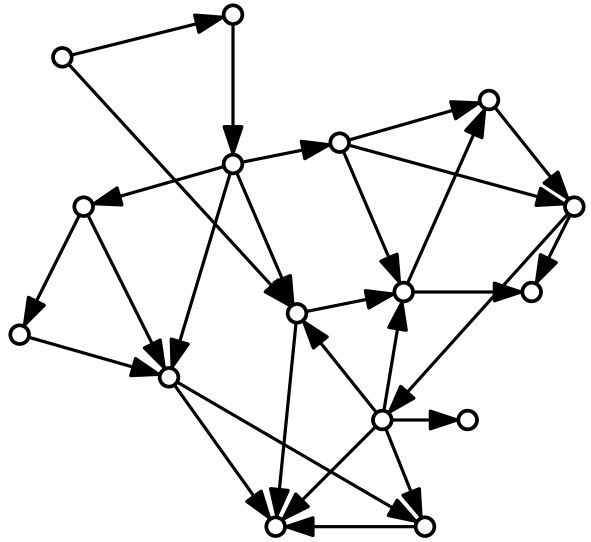


Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?



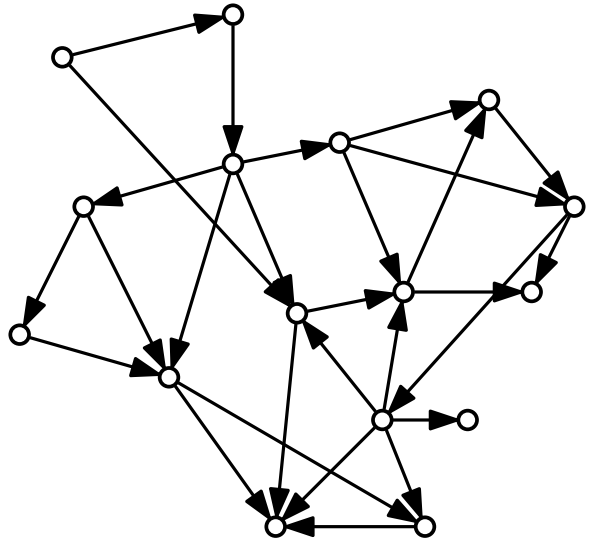
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



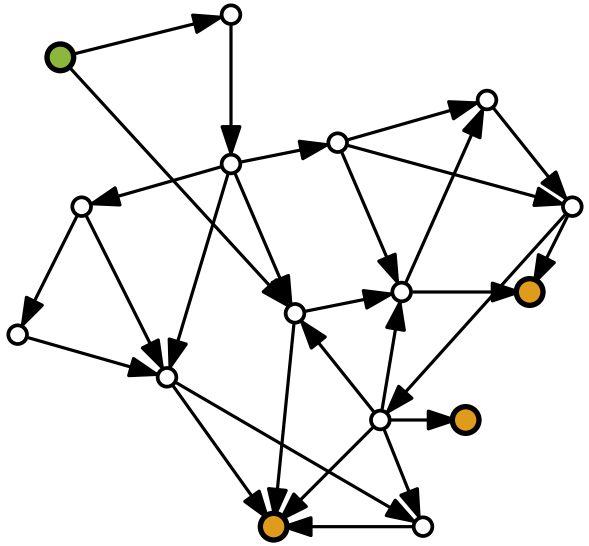
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



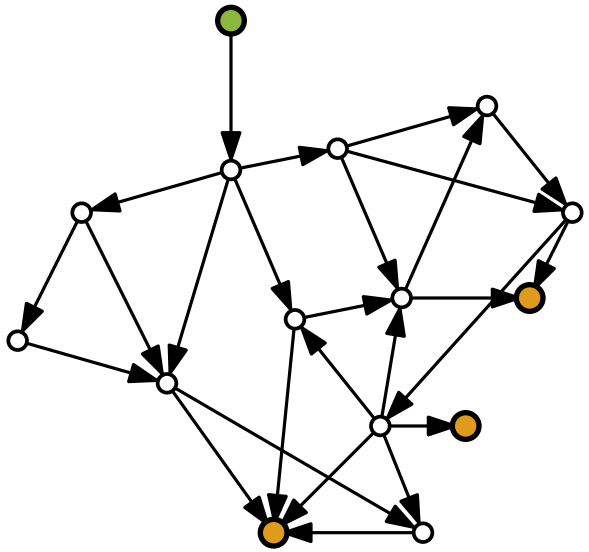
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



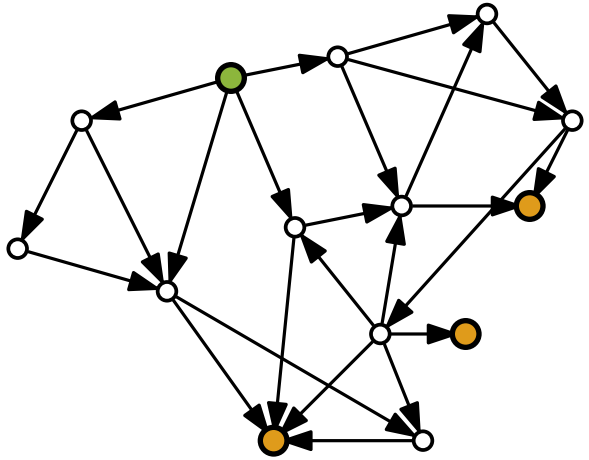
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



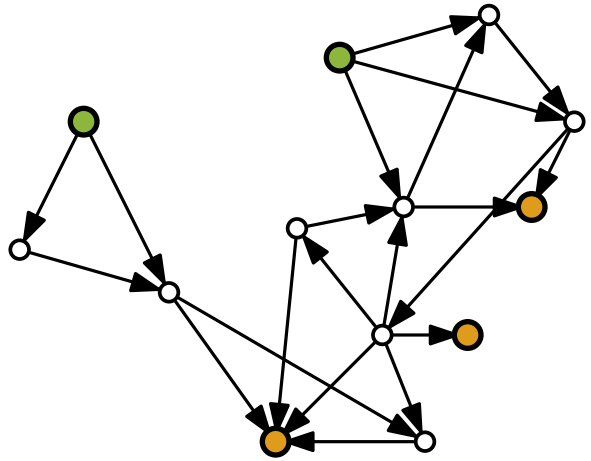
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke



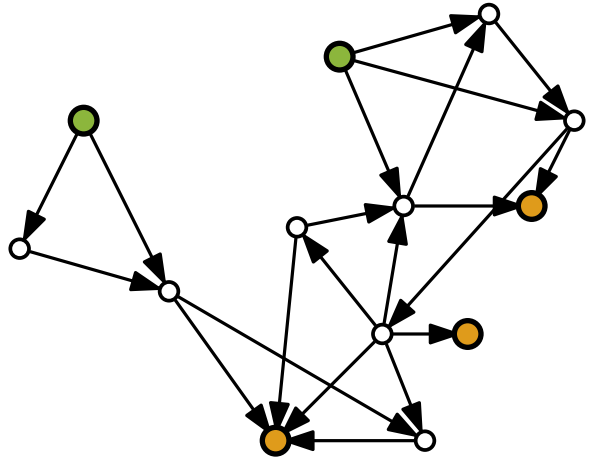
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)



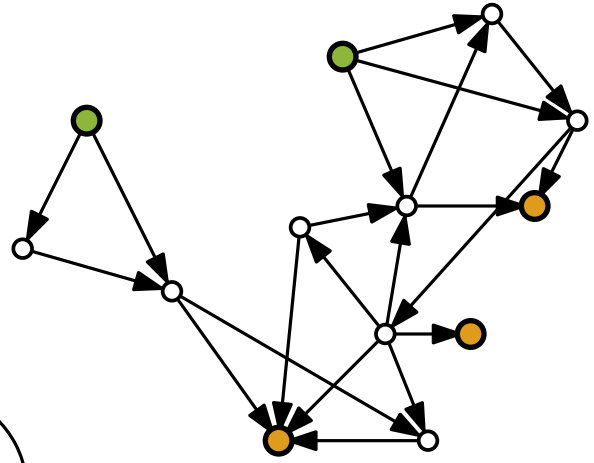
Cyclic Dependency

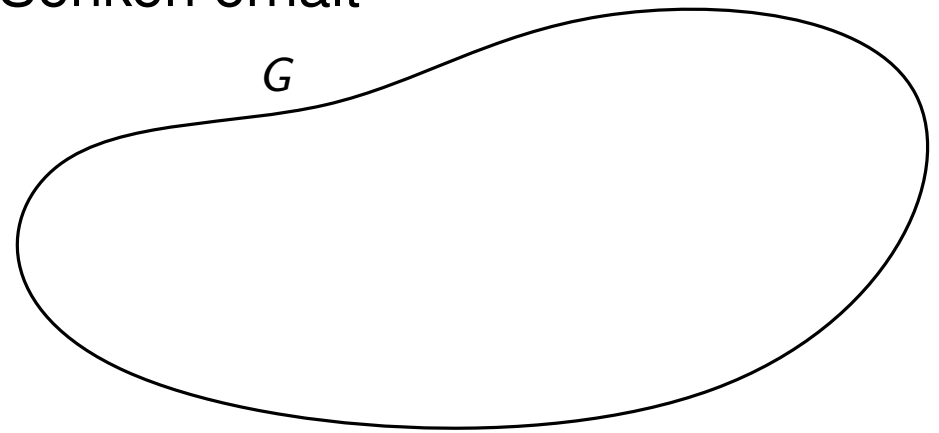
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)





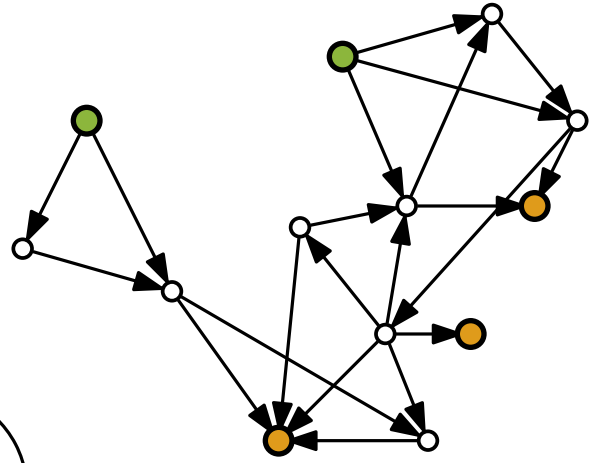
Cyclic Dependency

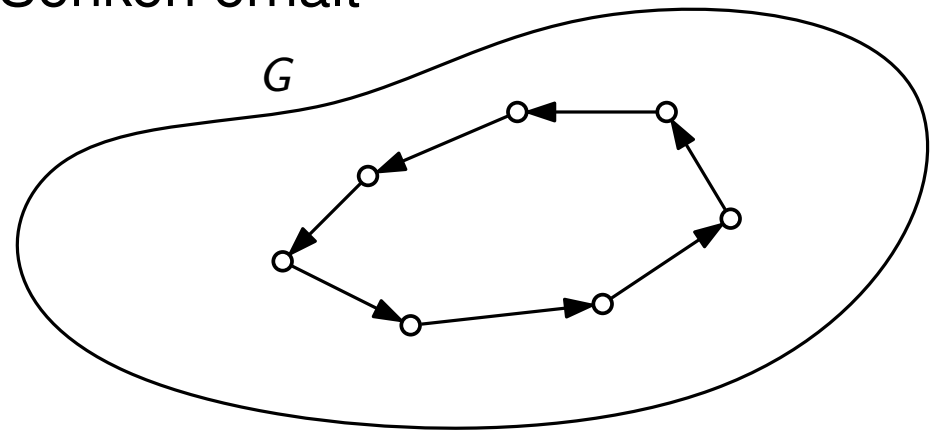
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)





Cyclic Dependency

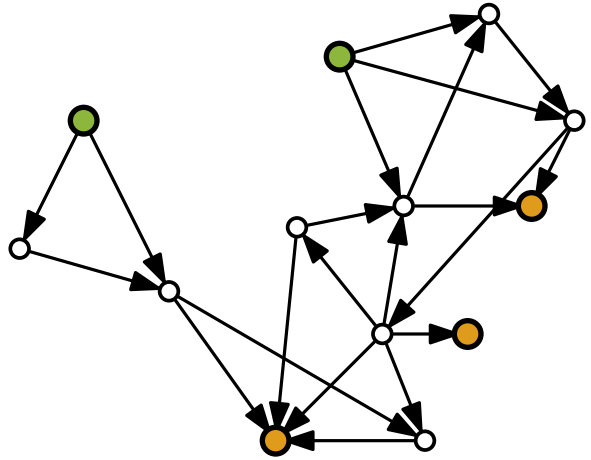
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung



Cyclic Dependency

Problemstellung

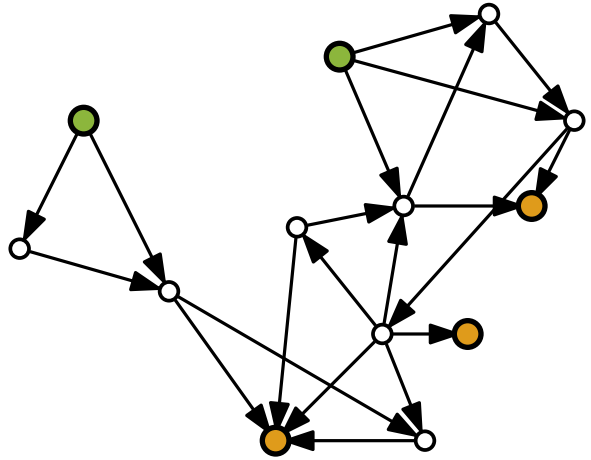
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen



Cyclic Dependency

Problemstellung

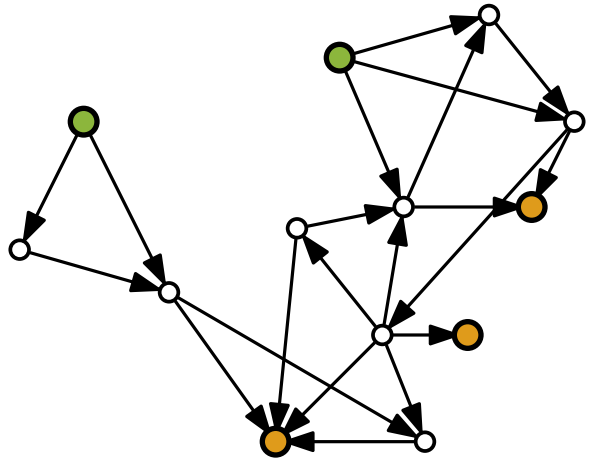
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$



Cyclic Dependency

Problemstellung

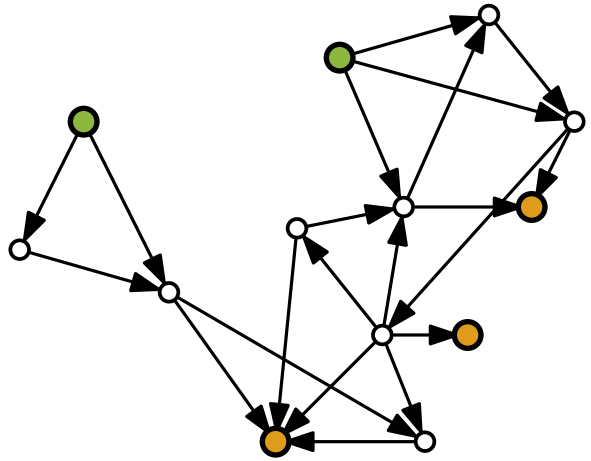
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q



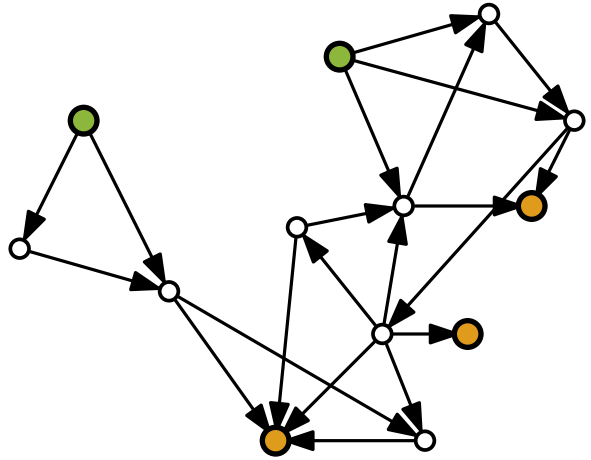
Cyclic Dependency

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)



Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q

Wie findet man neue Quellen?

Cyclic Dependency

Problemstellung

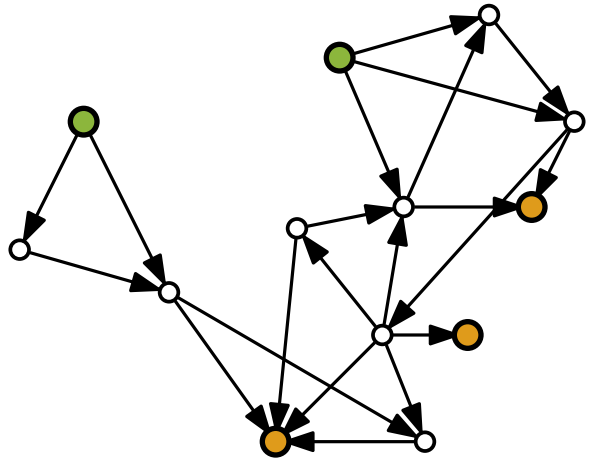
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



Cyclic Dependency

Problemstellung

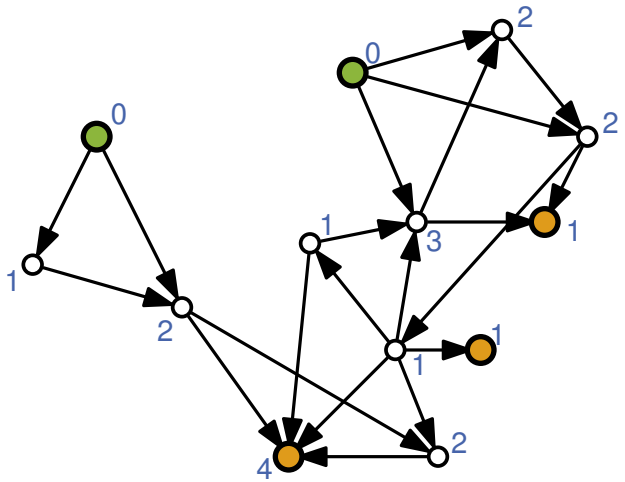
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



Cyclic Dependency

Problemstellung

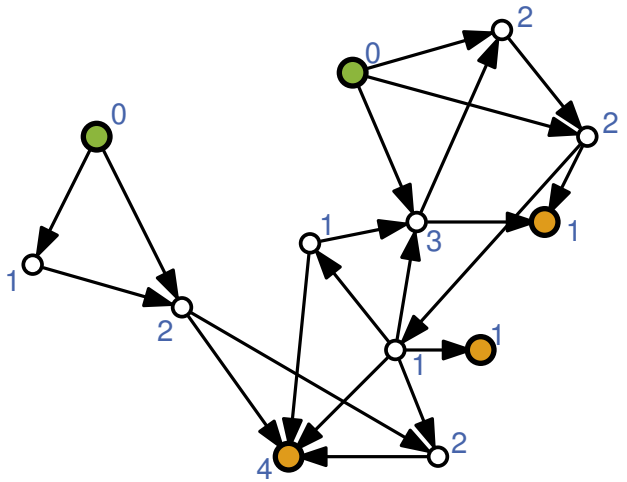
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

Cyclic Dependency

Problemstellung

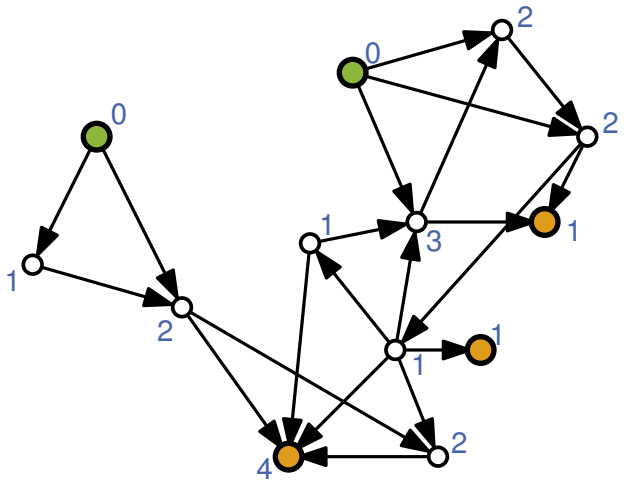
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

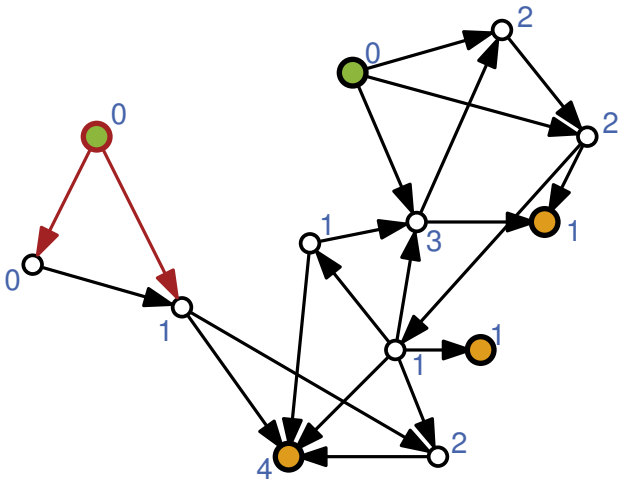
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 1, 0, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

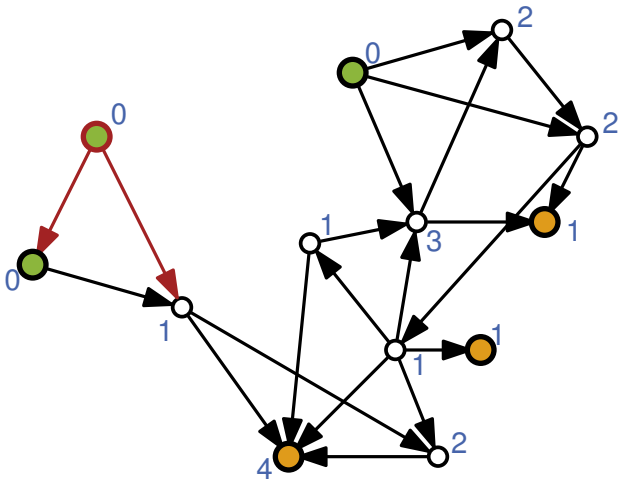
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 1, 0, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

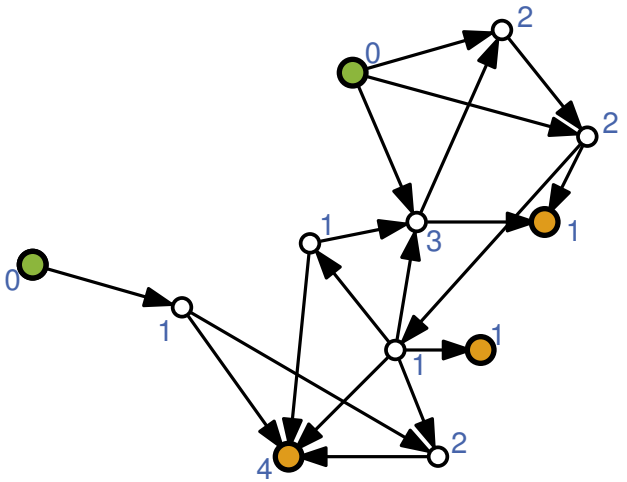
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 1, 0, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

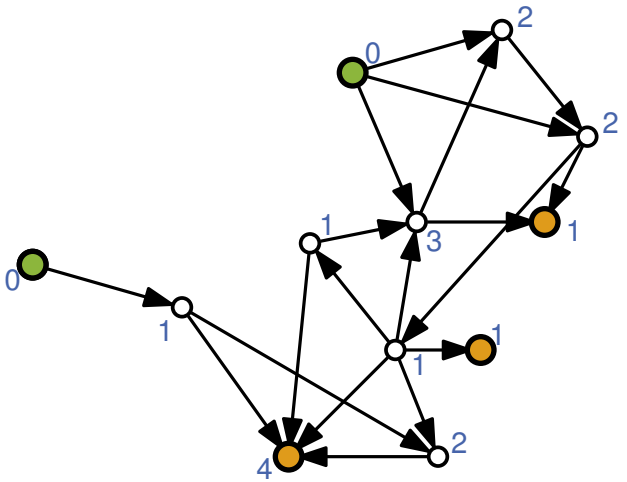
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 1, 0, \ldots]$

sources : Queue

Cyclic Dependency

Problemstellung

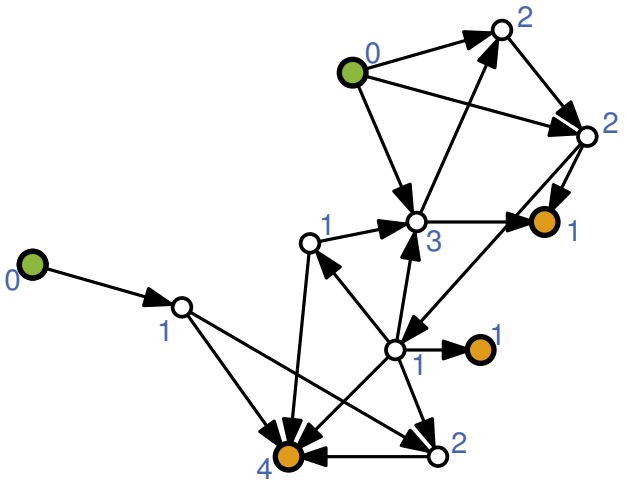
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Enthält G gerichteten Kreis?

Frage: Ist Linearzeit möglich?

- Beobachtung
  - DAG enthält immer Quelle, Senke
  - Löschen von Quellen/Senken erhält Kreis(freiheit)

Effiziente Implementierung

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$?
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 1, 0, \ldots]$

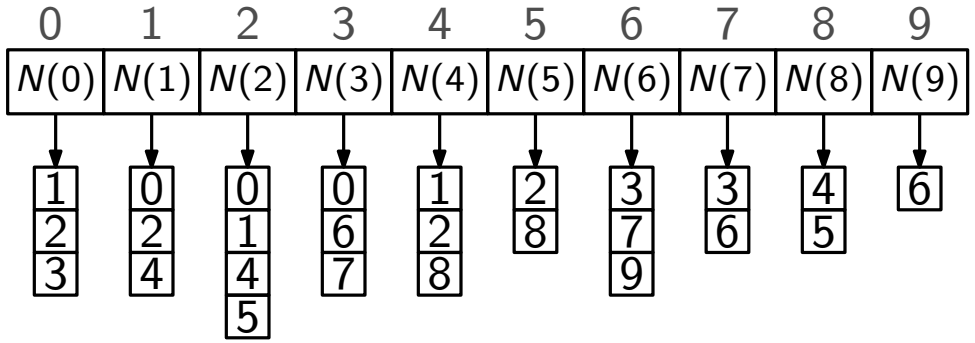
sources : Queue

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Knoten löschen

Adjazenzliste (ungerichteter Graph)

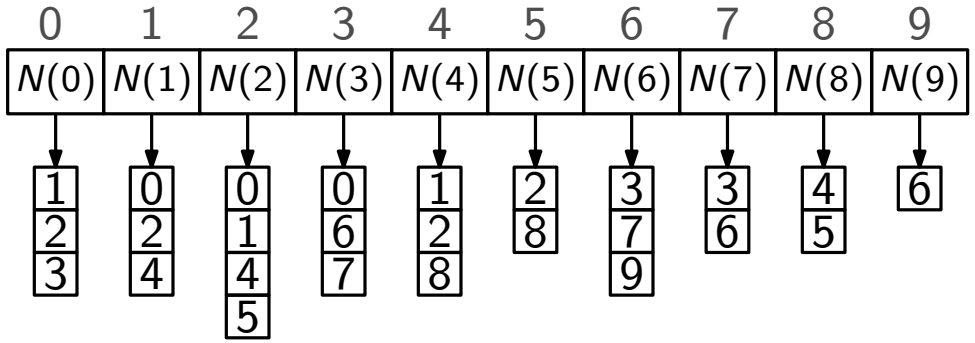


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten

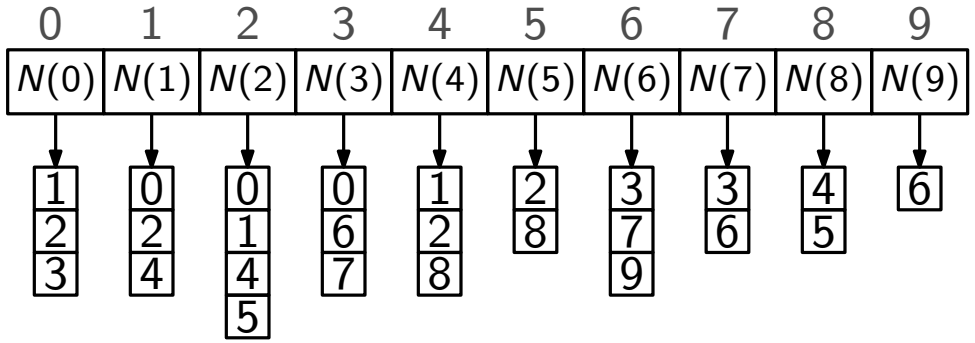


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



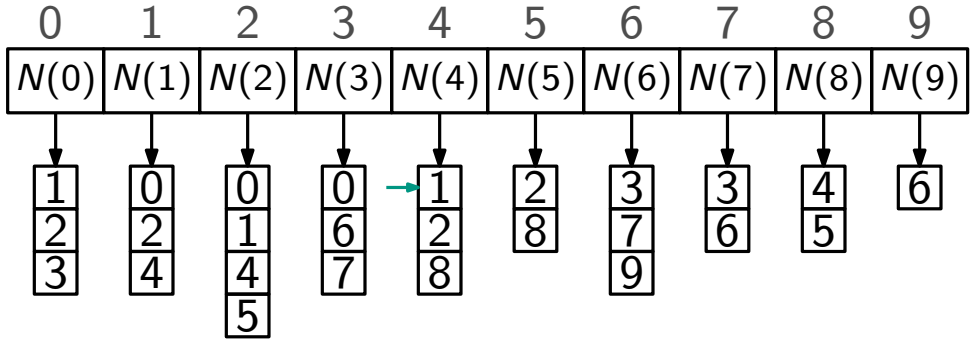
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



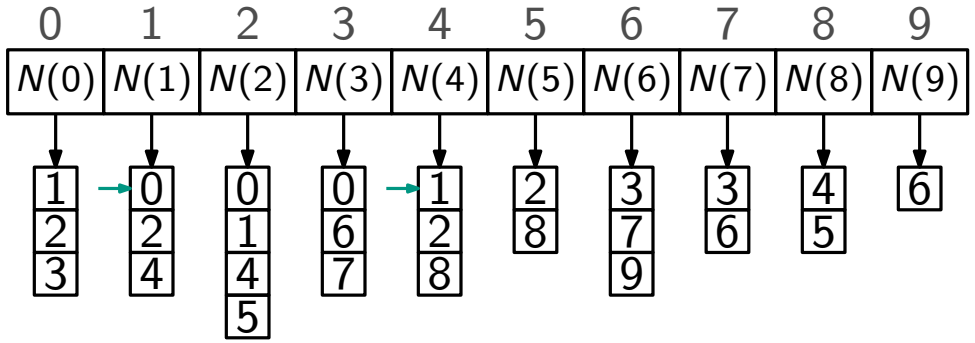
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



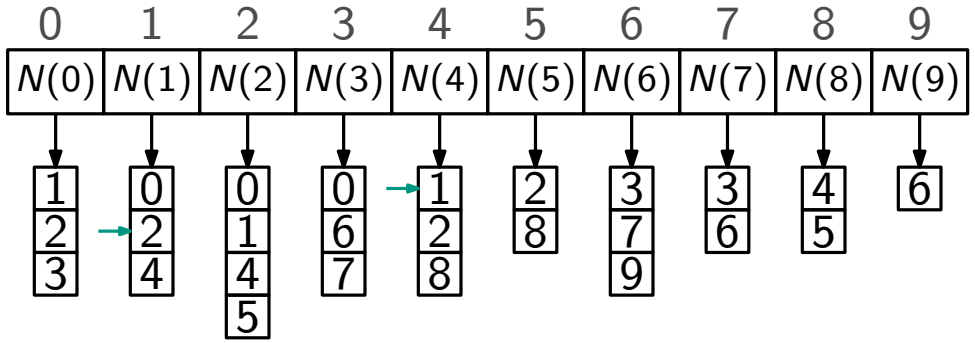
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



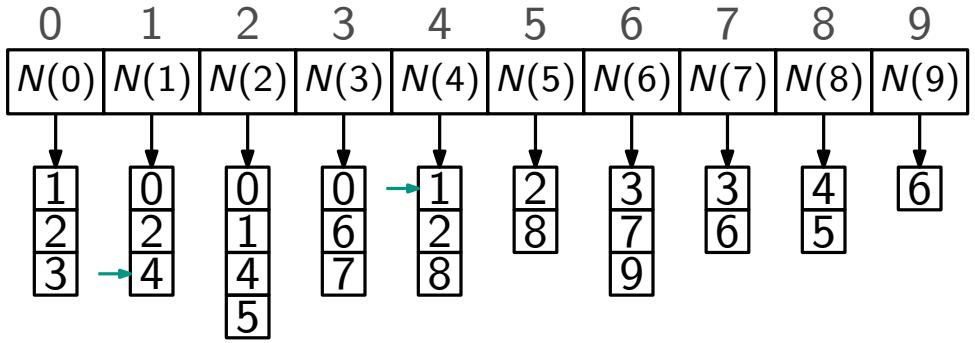
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



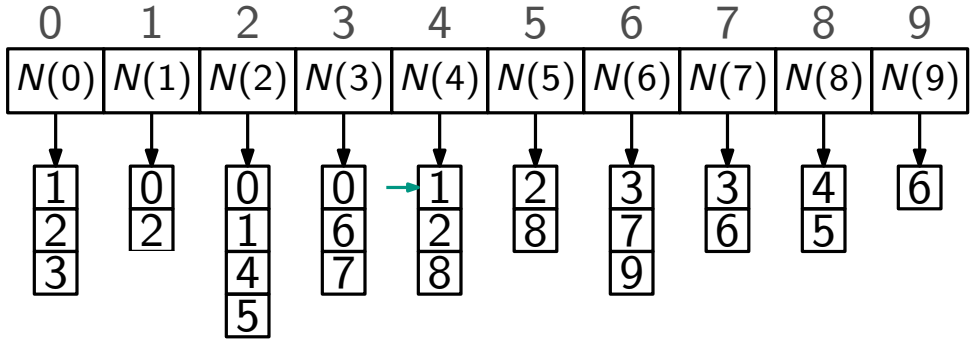
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



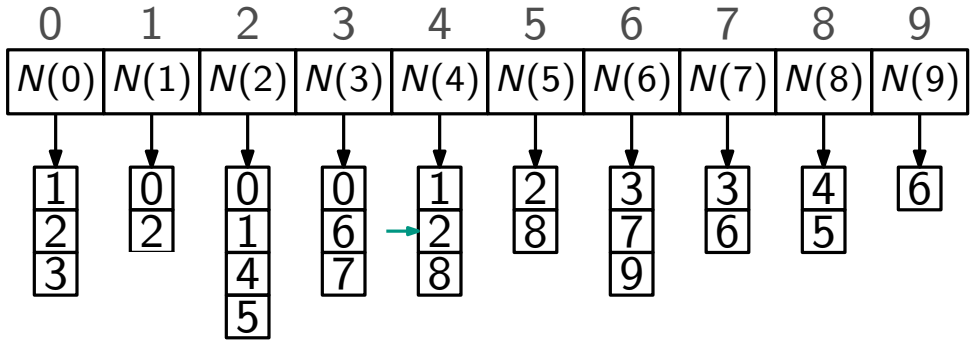
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



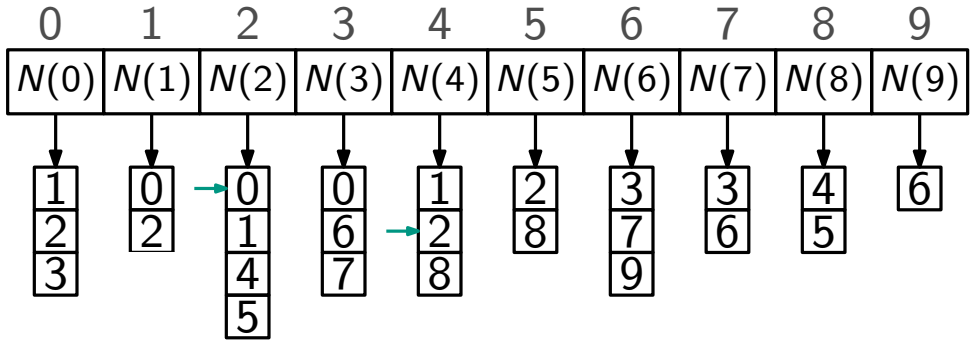
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



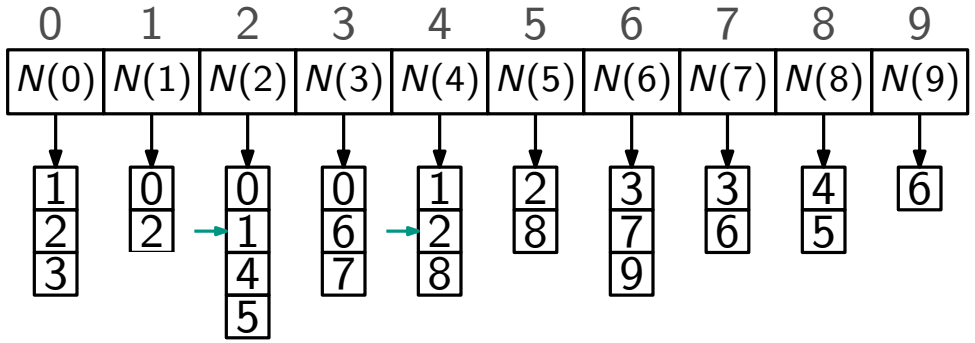
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



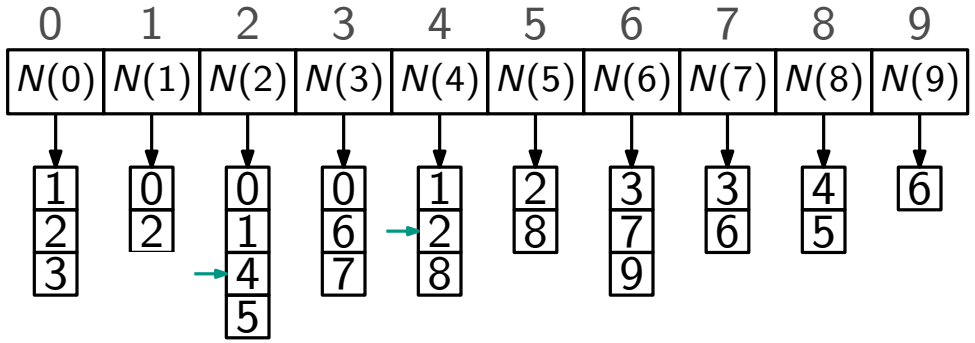
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



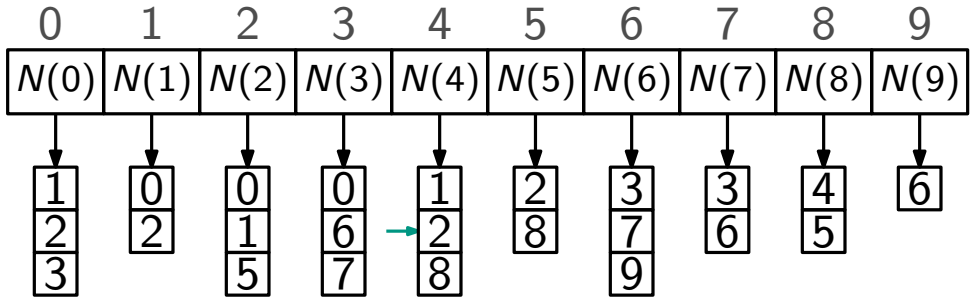
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



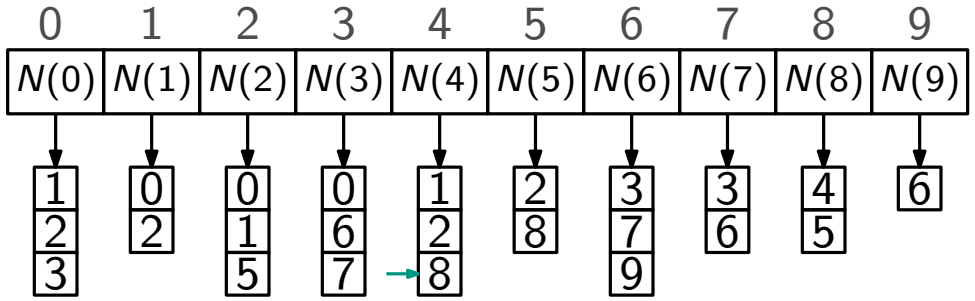
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



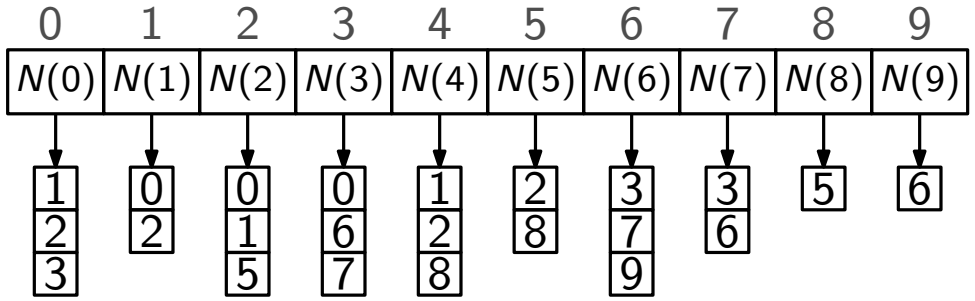
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



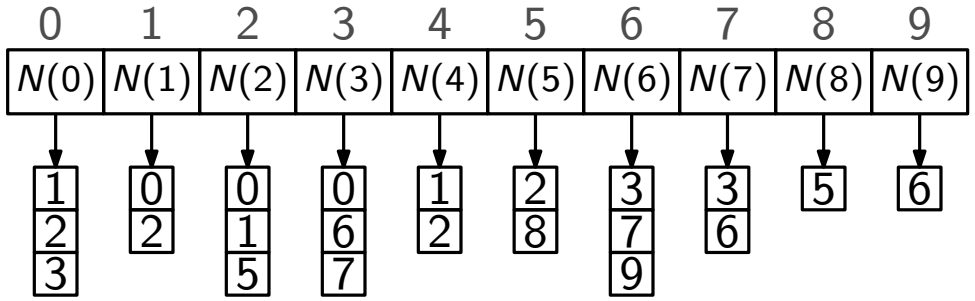
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



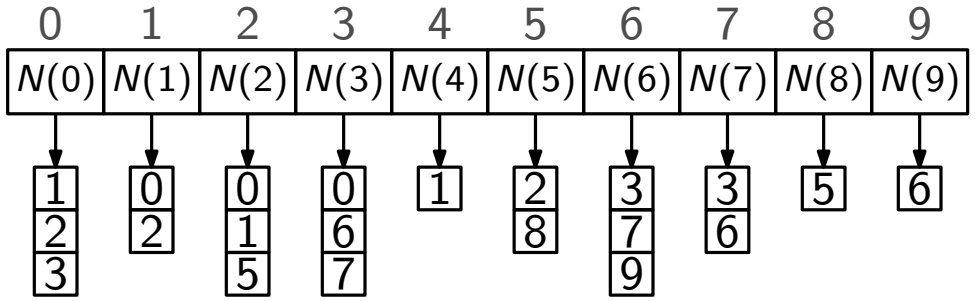
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



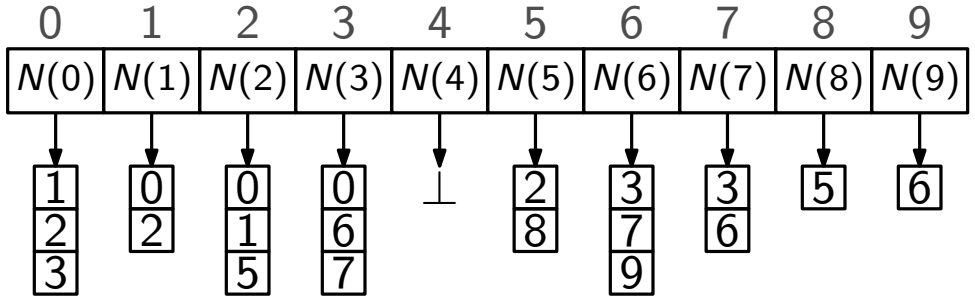
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten
- Kanten von gelöschten Knoten



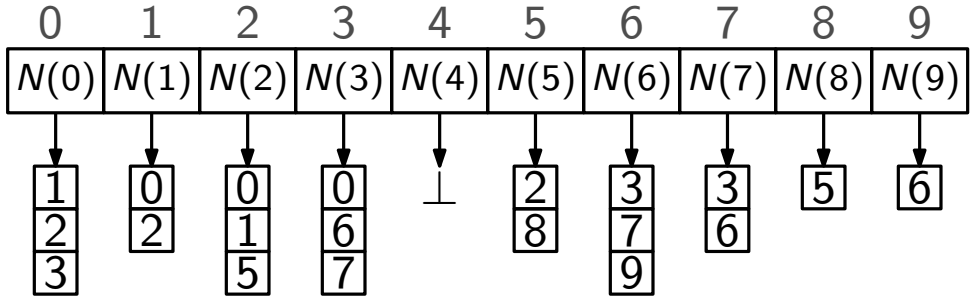
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten ✓
- Kanten von gelöschten Knoten ✓



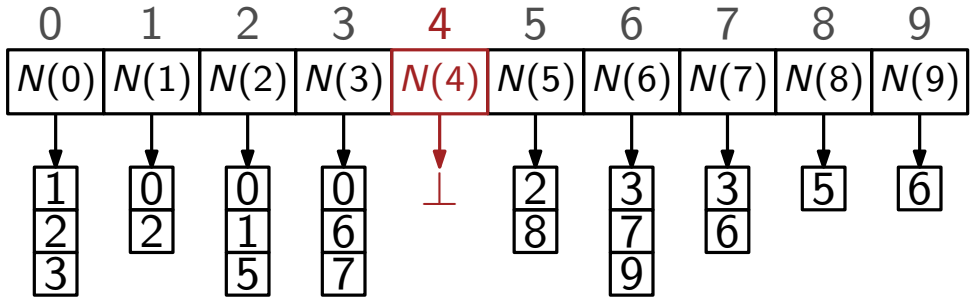
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten ✓
- Kanten von gelöschten Knoten ✓



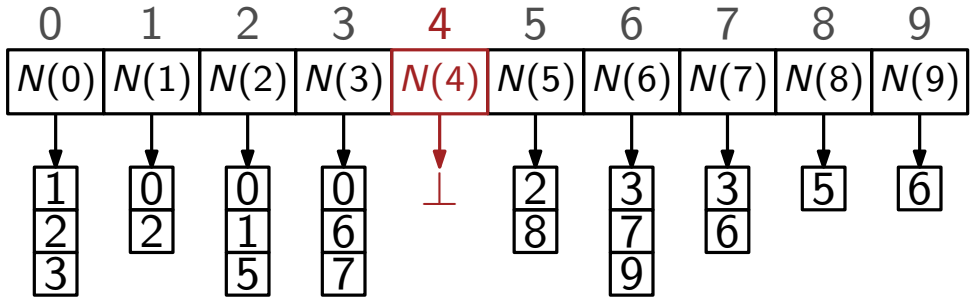
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Beispiel: Lösche 4

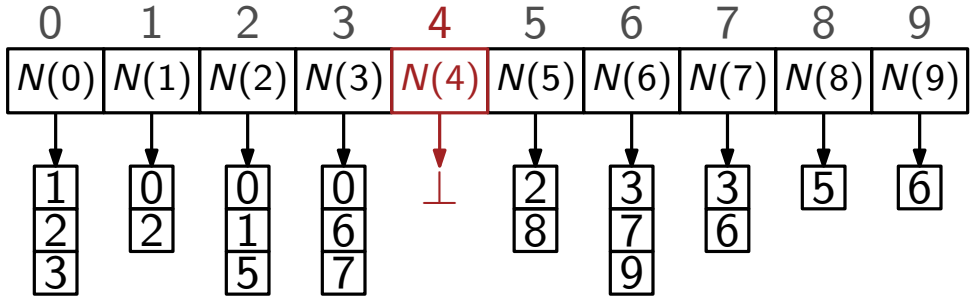
Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓

Besserer Ansatz



Beispiel: Lösche 4

Knoten löschen

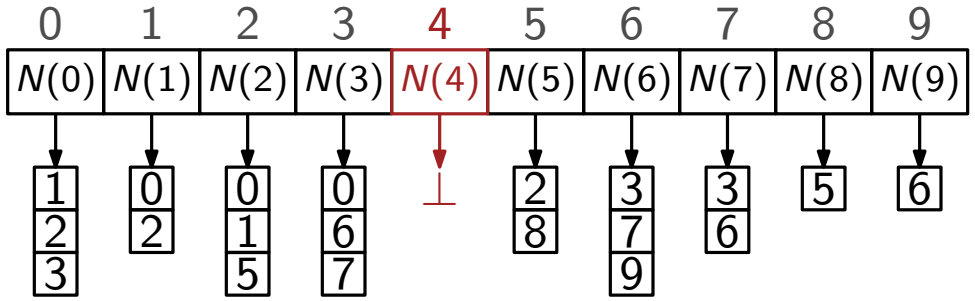
Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓

Besserer Ansatz

- speichere Pointer zwischen Listen



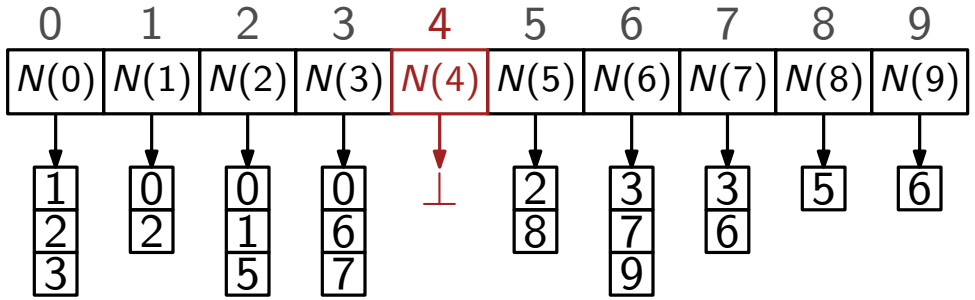
Beispiel: Lösche 4

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

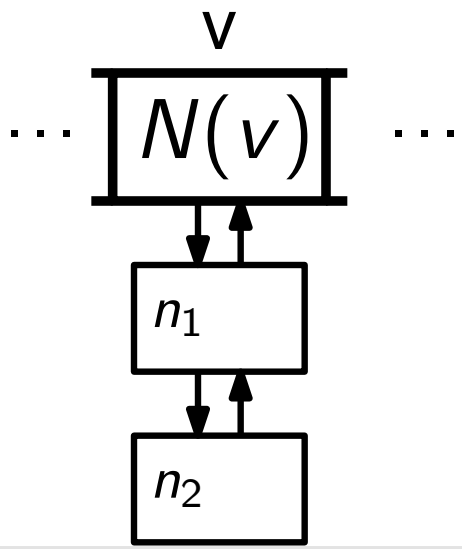
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen

Beispiel: Lösche 4

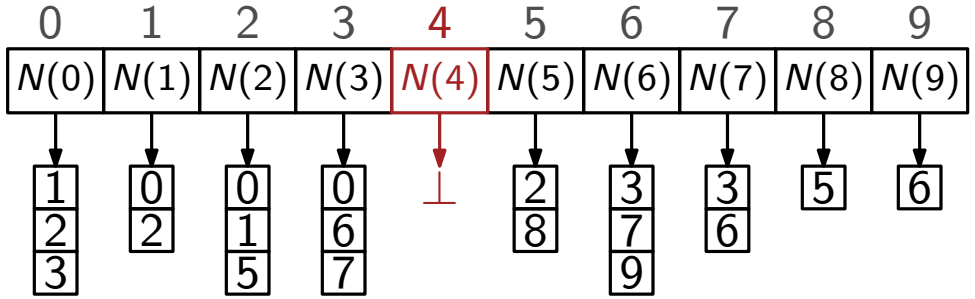


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

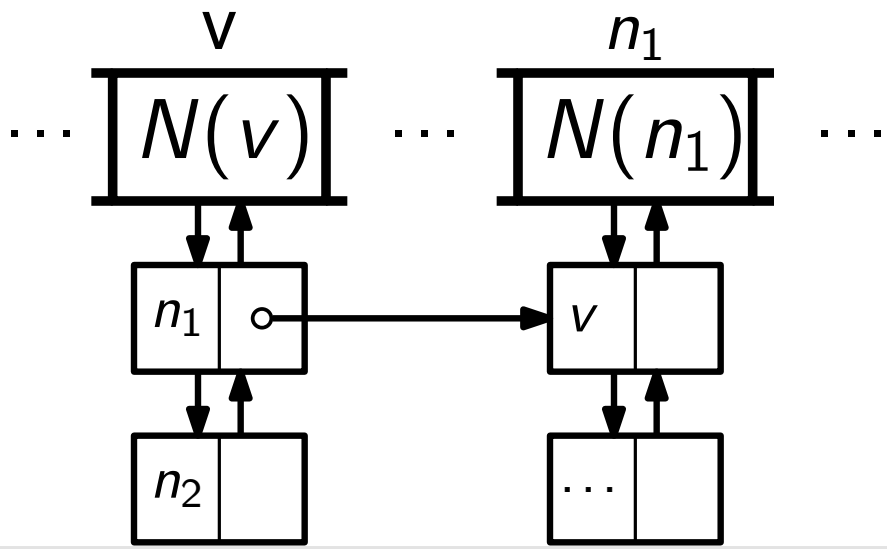
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen

Beispiel: Lösche 4

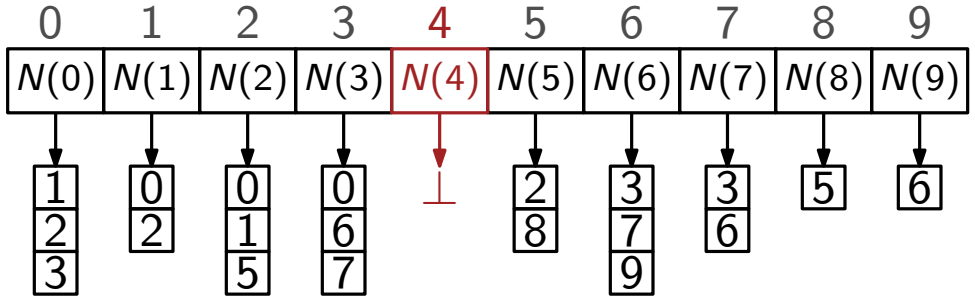


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

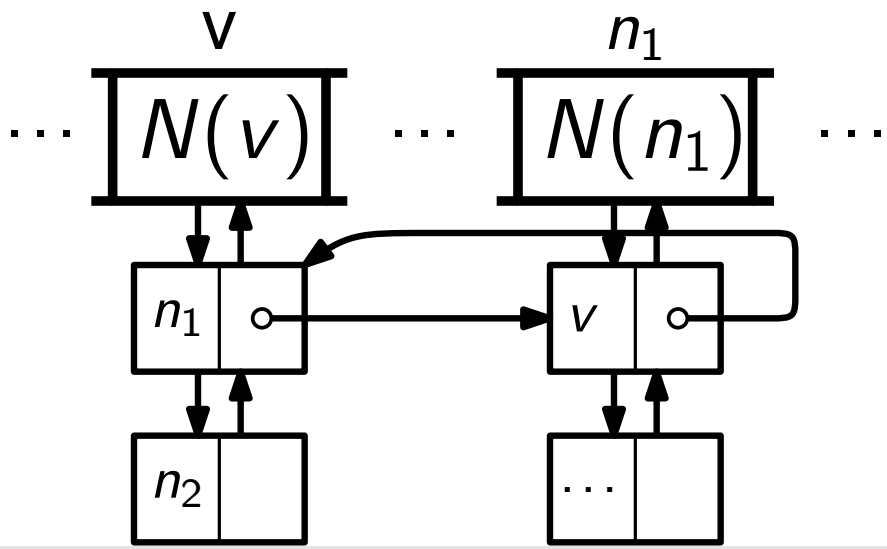
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen

Beispiel: Lösche 4

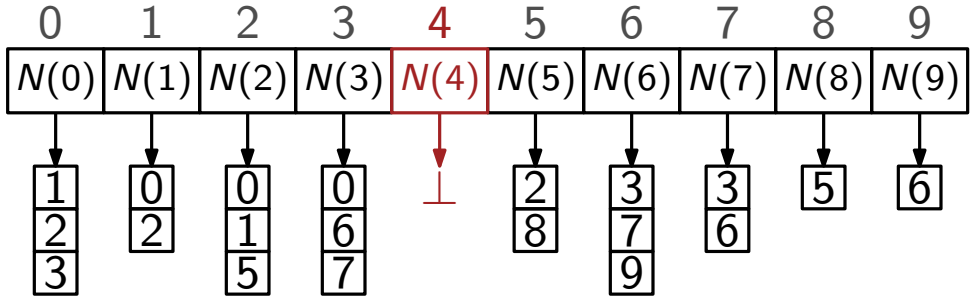


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

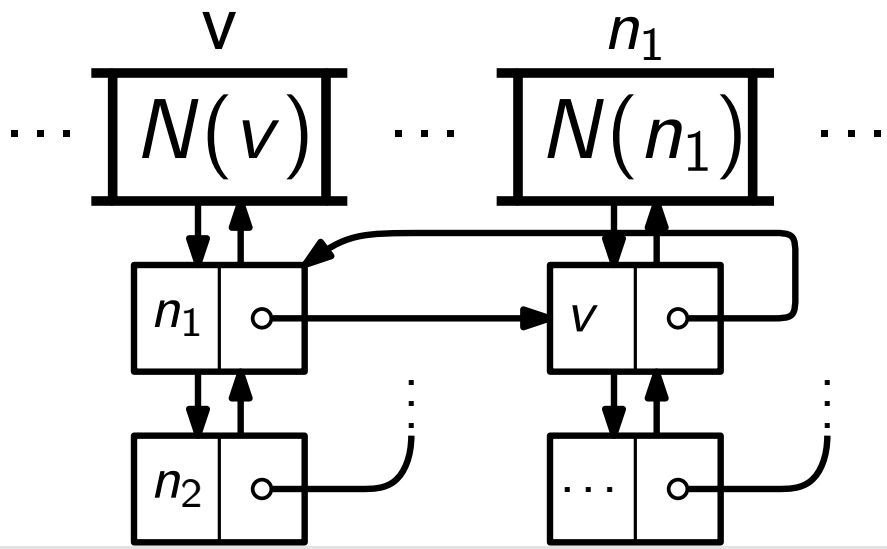
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen

Beispiel: Lösche 4

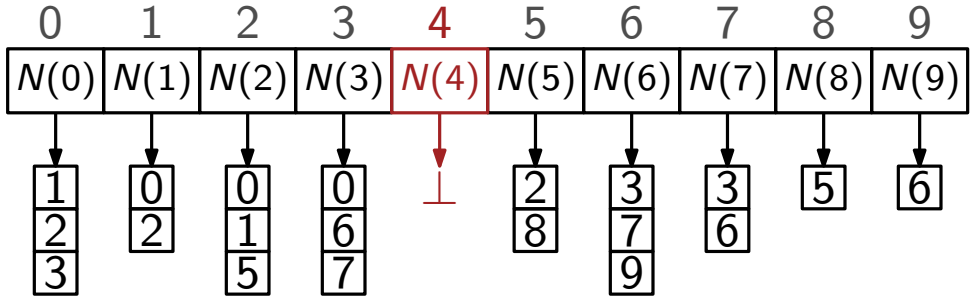


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

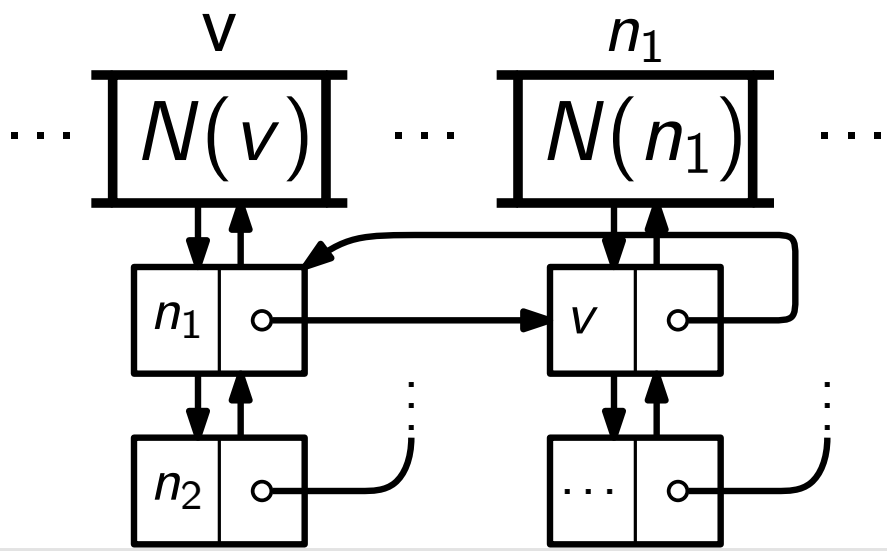
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$

Beispiel: Lösche 4

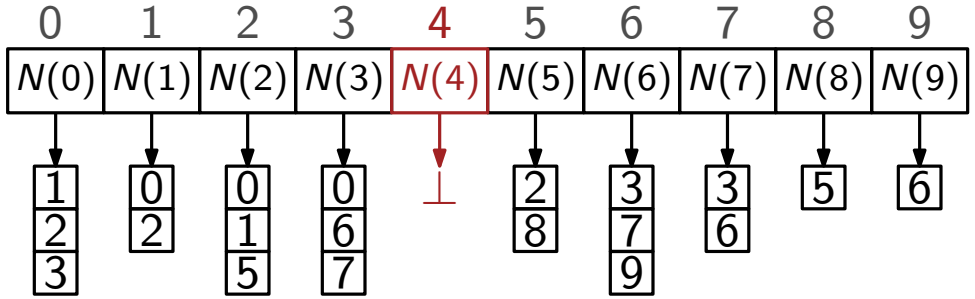


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

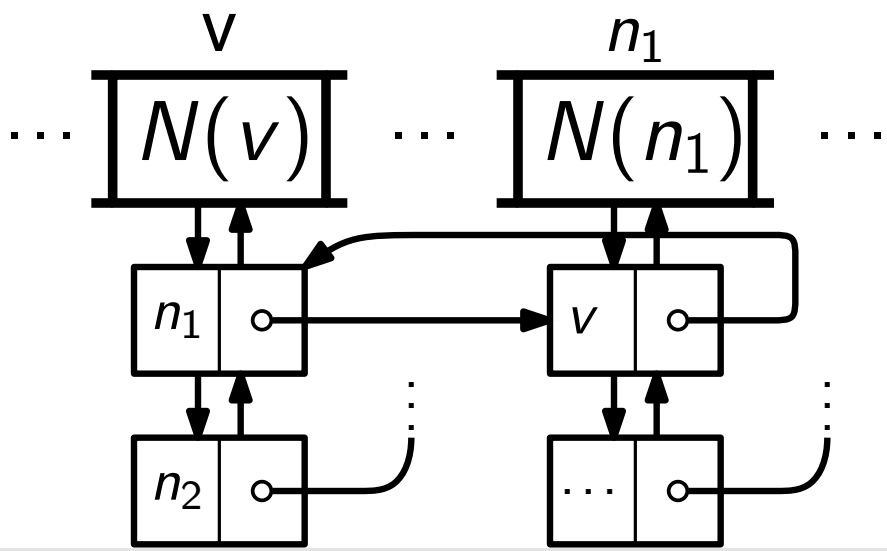
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$

Beispiel: Lösche 4

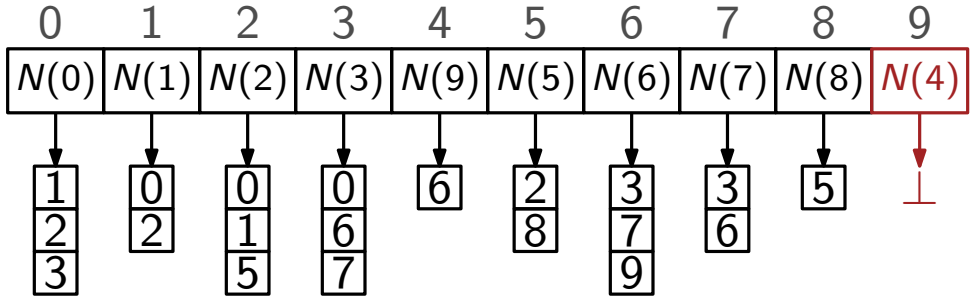


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

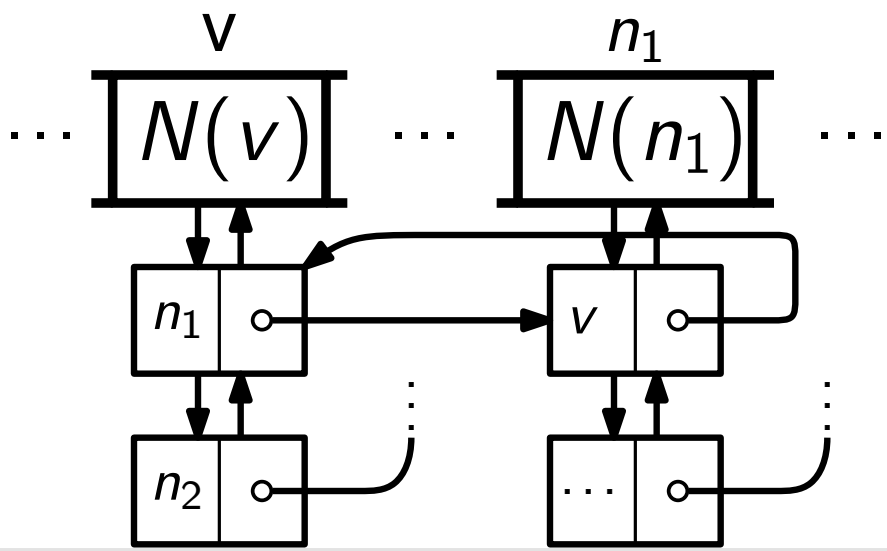
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$

Beispiel: Lösche 4

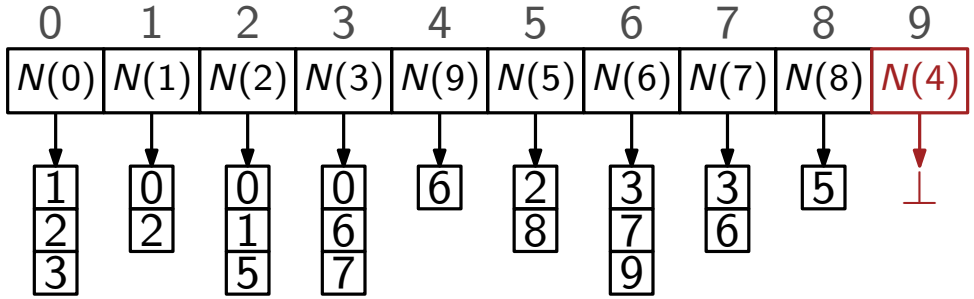


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

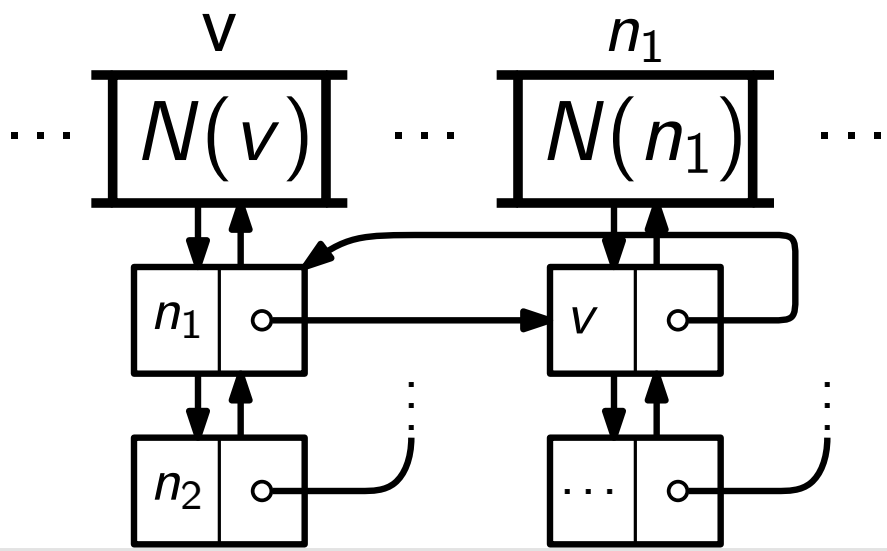
- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$

Beispiel: Lösche 4

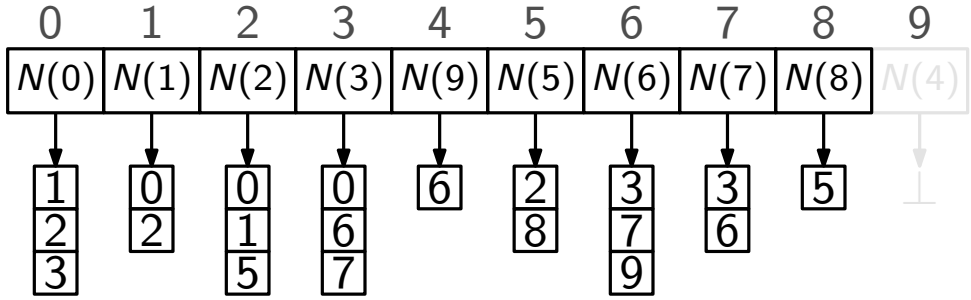


Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$

Beispiel: Lösche 4

Aktualisierung Indizes:

`new_index`[9] = 4

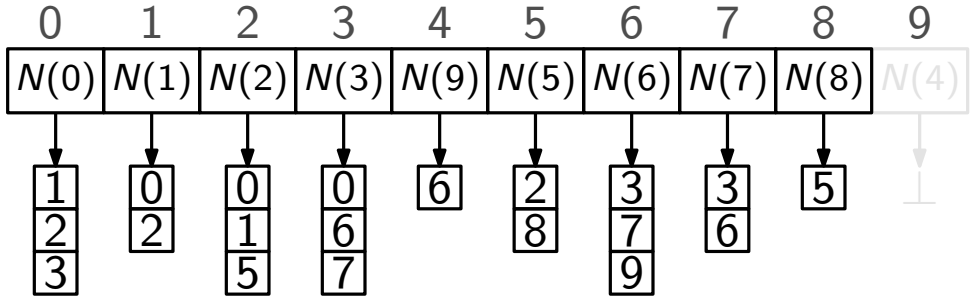
`old_index`[4] = 9

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$

Beispiel: Lösche 4

Aktualisierung Indizes:

`new_index`[9] = 4

`old_index`[4] = 9

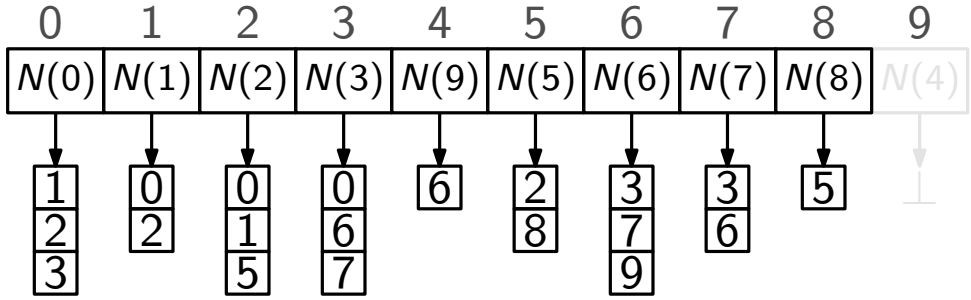
Gesamtlaufzeit: $\Theta(\deg(v))$

Knoten löschen

Adjazenzliste (ungerichteter Graph)

Naiver Ansatz: lösche

- Knoten
- Kanten zu gelöschten Knoten — teuer ✓
- Kanten von gelöschten Knoten ✓



Besserer Ansatz

- speichere Pointer zwischen Listen
  - lösche inzidente Kanten in $\Theta(\deg(v))$
- Knoten Löschen: swap mit Knoten $n-1$
  - mapping zw. neuen und alten Indizes
    `old_index`: $[\mathbb{N}]$, `new_index`: $[\mathbb{N}]$

Beispiel: Lösche 4

Aktualisierung Indizes:

`new_index`[9] = 4

`old_index`[4] = 9

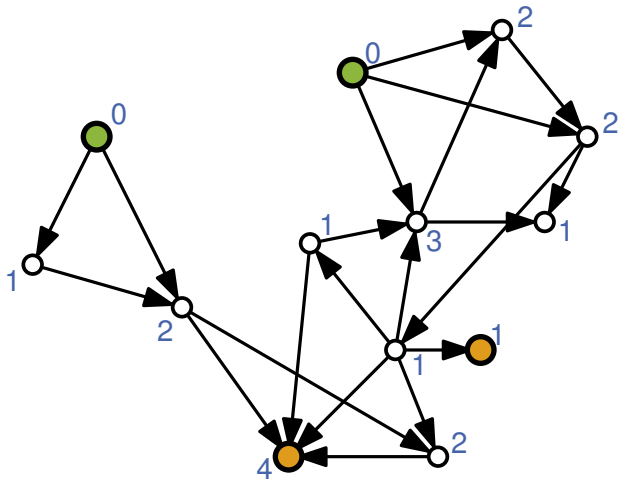
Gesamtlaufzeit: $\Theta(\deg(v))$

Und auf gerichteten Graphen?

Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?



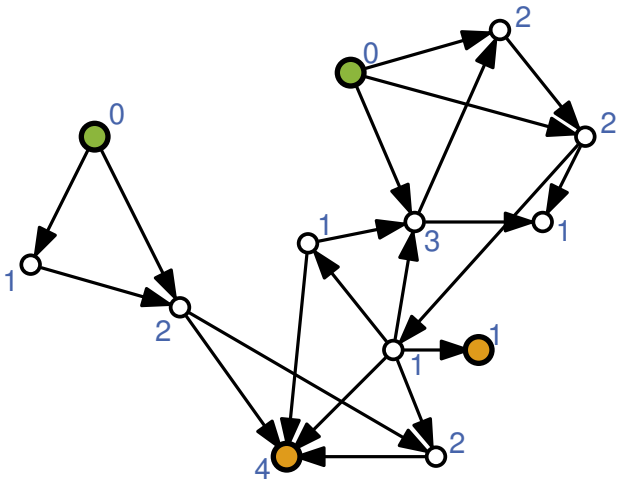
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$?
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

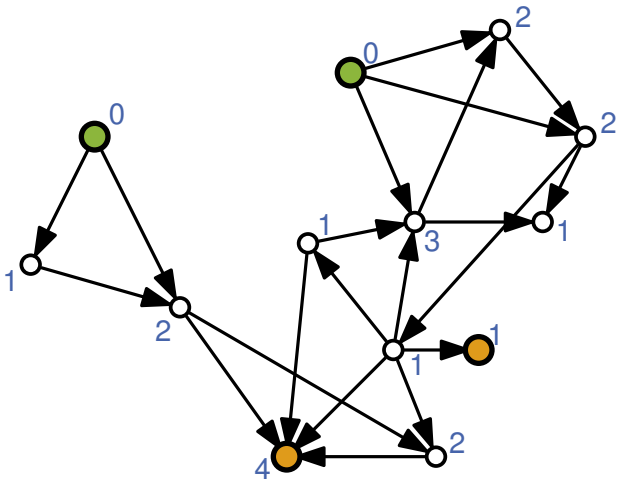
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

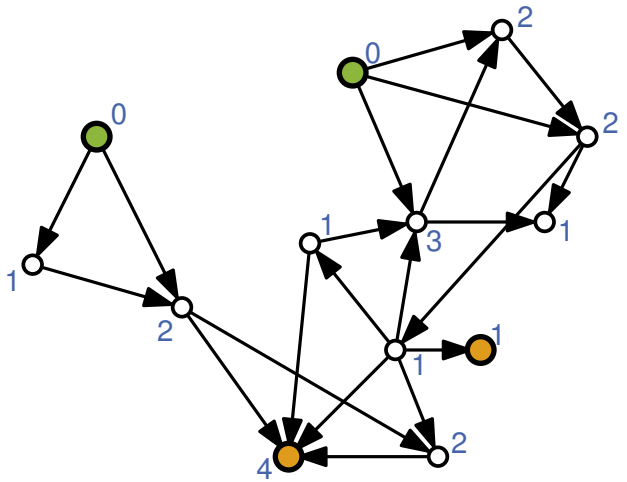
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(\deg_{\text{out}}(v))$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

deleted : $[\mathbb{N}]$

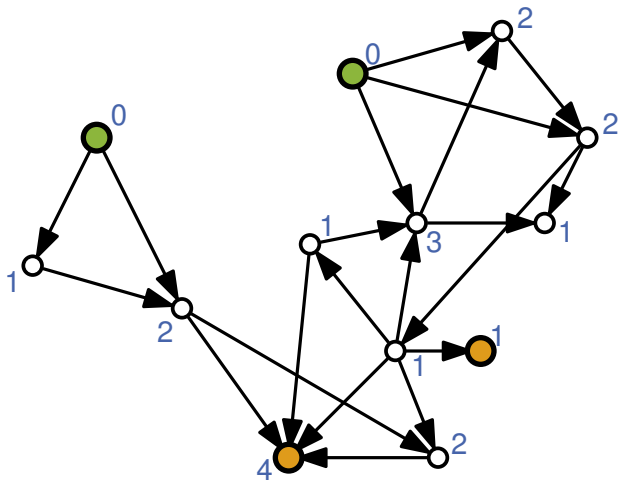
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

deleted : $[\mathbb{N}]$

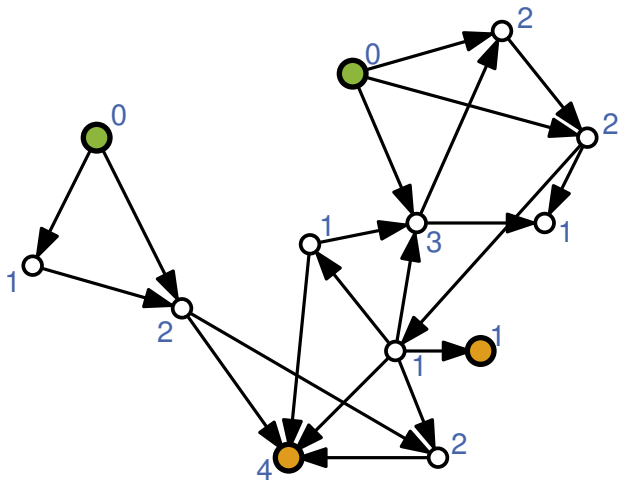
Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

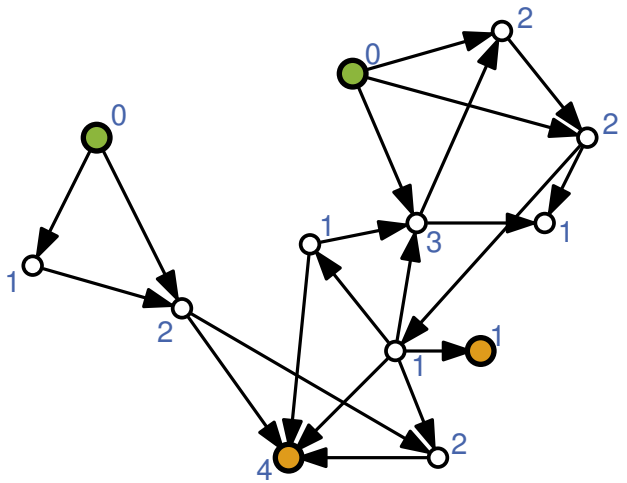
Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

Problemstellung

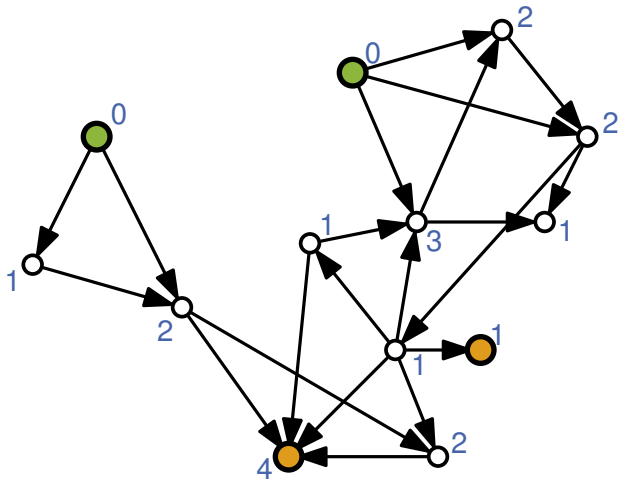
- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$

Wichtig



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

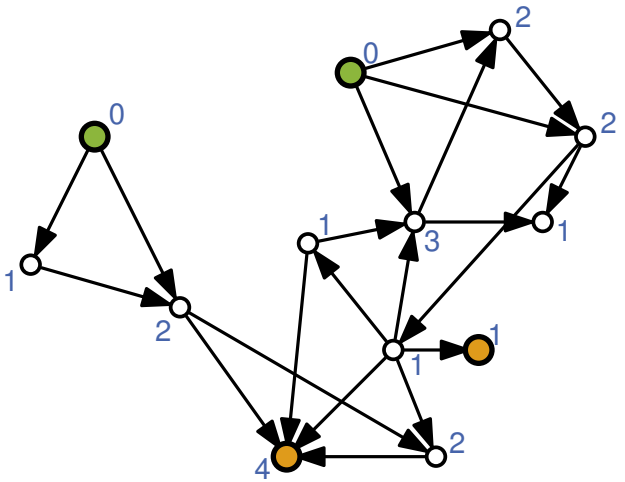
Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$

Wichtig

- Knoten *löschen* oft nicht notwendig



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue

Knoten löschen?

Problemstellung

- gegeben: gerichteter Graph $G = (V, E)$
- Frage: Ist G DAG?

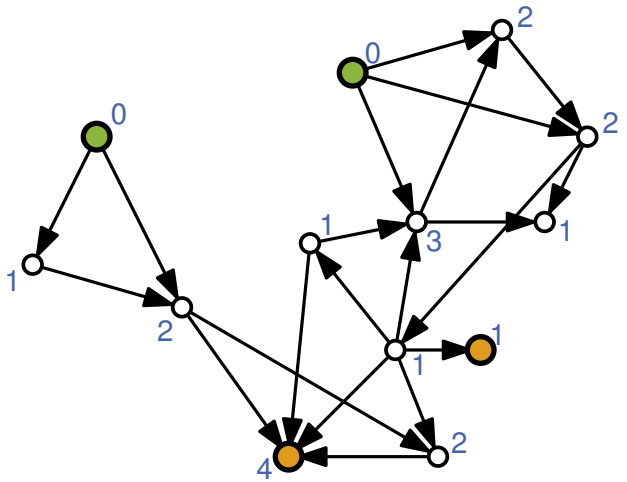
Algorithmus

- finde zu Beginn alle Quellen $\Theta(n + m)$
- lösche iterativ Quelle q $\Theta(1)$
  - suche neue Quellen in $N(q)$ $\Theta(\deg_{\text{out}}(v))$

Gesamtlaufzeit: $\Theta(n + m)$

Wichtig

- Knoten *löschen* oft nicht notwendig
- geschicktes Verwalten zusätzlicher Informationen hilfreich



$\deg_{\text{in}} : [\mathbb{N}] = [0, 0, 2, 1, \ldots]$

sources : Queue