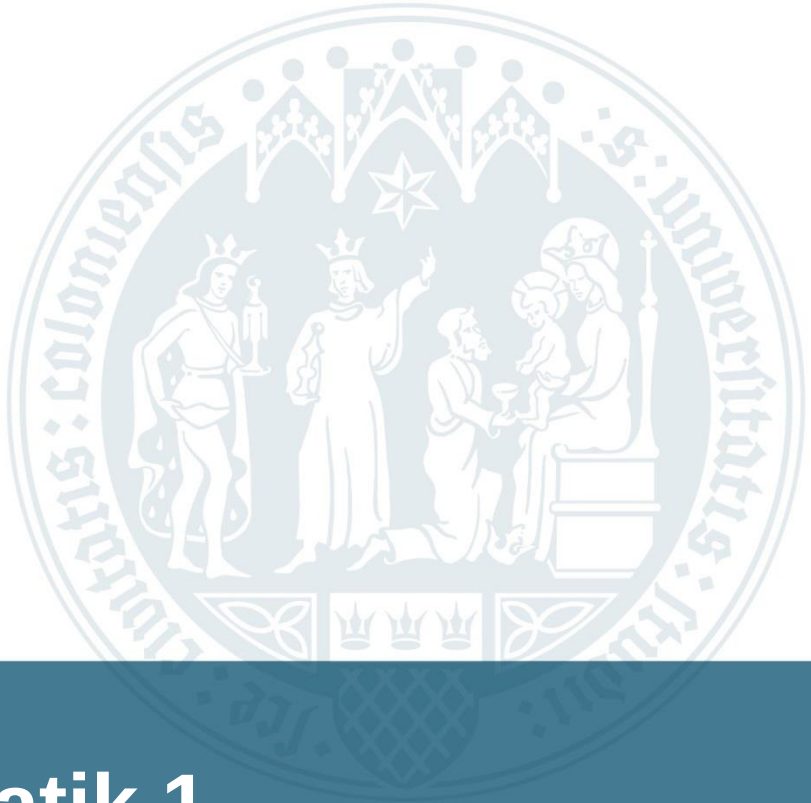
Universität zu Köln



Grundzüge der Informatik 1

Vorlesung 19

Überblick Vorlesung

- Direkte Adressierung
- Hash-Tabellen
  - Auflösen von Kollisionen
  - Wahl der Hash-Funktion
  - Offene Adressierung
- Graphalgorithmen
  - Grundlegende Begriffe der Graphentheorie
  - Datenstrukturen zum Speichern von Graphen
  - Beginn: Kürzeste Wege in ungewichteten Graphen

Datenstrukturen

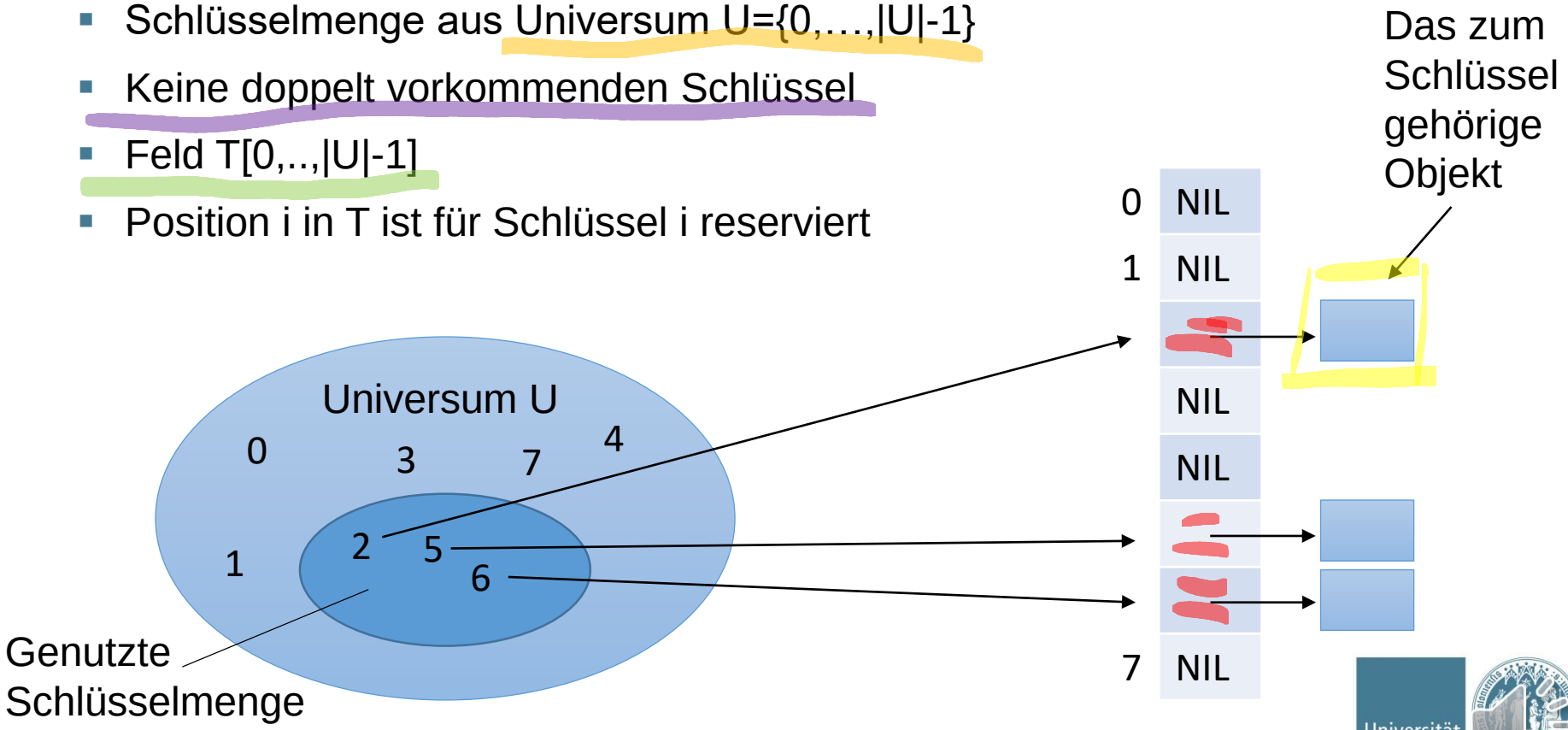
Frage

- Gibt es effizientere Datenstrukturen für unser Datenverwaltungsproblem als Rot-Schwarz-Bäume?

Datenstrukturen

Felder mit direkter Adressierung

- Schlüsselmenge aus Universum U={0,…,|U|-1}
- Keine doppelt vorkommenden Schlüssel
- Feld T[0,..,|U|-1]
- Position i in T ist für Schlüssel i reserviert



Datenstrukturen

DirectAddressSearch(k)

1. **return** T[k]

DirectAddressInsert(x)

1. T[key[x]] = x

DirectAddressDelete(k)

1. T[k] = NIL

Datenstrukturen

Direkte Adressierung

- Suchen, Einfügen und Löschen in O(1) Zeit
- Schlüssel müssen aus bekanntem Universum U={0,…,|U|-1} stammen
- Speicherbedarf $\Omega(|U|)$

Datenstrukturen

Hash-Tabellen

- Speicherbedarf von direkter Adressierung ist unrealistisch und ineffizient
- Wir nutzen Hashfunktion h, die Universum U auf eine Hash-Tabelle T[0…m-1] abbildet
- $h: U \rightarrow \{0,\ldots,m-1\}$
- Für einen Schlüssel k nennen wir h(k) den Hash-Wert von k

Datenstrukturen

Hash-Tabellen

- Speicherbedarf von direkter Adressierung ist unrealistisch und ineffizient
- Wir nutzen Hashfunktion h, die Universum U auf eine Hash-Tabelle T[0…m-1] abbildet
- h: U $\rightarrow$ {0,…,m-1}
- Für einen Schlüssel k nennen wir h(k) den Hash-Wert von k

Problem

- Es kann sein, dass mehrere Schlüssel aus der Schlüsselmenge denselben Hash-Wert haben (Kollision)
- Hash-Tabellen mit Verkettung: Jede Zelle der Hash-Tabellen enthält einen Zeiger auf eine Liste; Die Schlüssel mit Hash-Wert i werden in der Liste T[i] abgespeichert

Datenstrukturen

Einfaches Beispiel

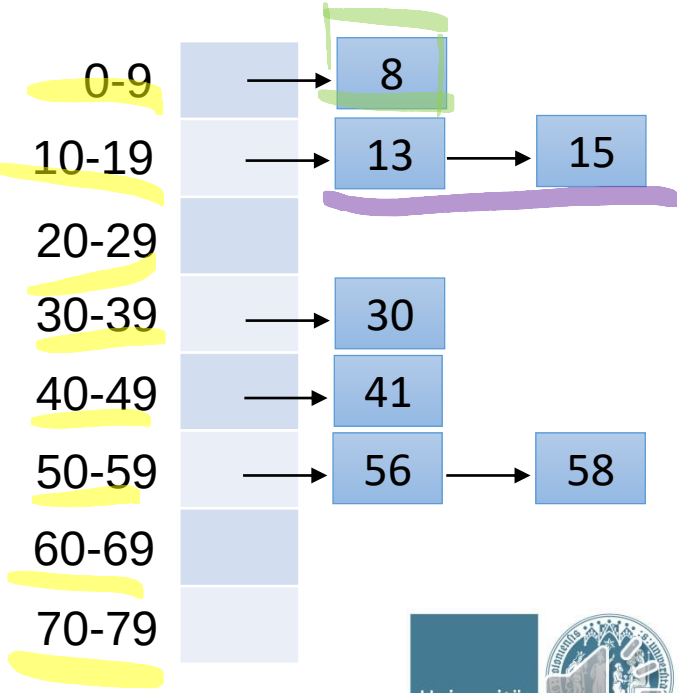
- $h(k) = \lfloor k/10 \rfloor$

Beispiel

- Schlüsselmenge aus Universum {0,..,79}
  8, 13, 15, 30, 41, 56, 58

Problem

- 13, 15 und 56, 58 liegen im selben Bereich



Datenstrukturen

ChainedHashSearch(k)

1. Suche nach Schlüssel k in Liste T[h(k)]

ChainedHashInsert(k)

1. Füge k am Kopf der Liste T[h(k)] ein

Chained HashDelete(k)

1. Lösche k aus der Liste T[h(k)]

Datenstrukturen

Wahl der Hash-Funktion

- Idee: Wähle die Funktion zufällig
- Beispiel: Wenn jeder Schlüssel auf einen zufälligen Hash-Wert abgebildet wird, erwarten wir wenig Kollisionen
- Leider benötigt die Speicherung einer vollständig zufälligen Hashfunktion viel Speicher
- Abhilfe: Wähle Hashfunktion aus einer geeigneten Menge zufällig, so dass sich die Hash-Funktion ähnlich wie eine vollständig zufällige Hashfunktion verhält

Datenstrukturen

Wahl von Hash-Funktionen

- Die Divisionsmethode
  - Definiere $h(k) = k \bmod m$
  - Häufig wählt man m als Primzahl nicht zu nah an einer Zweierpotenz
- Die Multiplikationsmethode
  - $h(k) = \lfloor m\,(kA - \lfloor kA \rfloor) \rfloor$ für $0<A<1$
  - Wahl von m unkritisch
- Universelles Hashing
  - $h_{a,b}(k) = ((ak+b) \bmod p) \bmod m$
  - Wähle a zufällig aus {1,…,p-1} und b aus {0,…,p-1}
- Analyse nicht Stoff dieser Vorlesung!

Offene Adressierung mit linearem Ausprobieren

- Alle Schlüssel werden in der Hash-Tabelle selber gespeichert
- Versuche zunächst, den Schlüssel in T[h(k)] einzufügen, wenn das nicht geht, in T[h(k)+1], T[h(k)+2], usw. bis ein Platz gefunden wurde. Dabei werden die Indizes modulo m genommen.
- Beim Suchen wird auf die gleiche Weise vorgegangen. Die Suche hört auf, wenn der Schlüssel gefunden wird oder eine leere Zelle oder alle Zellen durchgegangen wurden

Datenstrukturen

Markierung leerer Zellen

- Wir verwenden -1 (oder allgemeiner einen Wert, der kein Schlüssel ist), um eine leere Zelle zu markieren
- Annahme: Zu Beginn ist die Tabelle mit -1 ausgefüllt

Datenstrukturen

Hash-Einfügen(T,k**)**

1. i=0
2. **while** i<m **do**
3. j=(h(k)+i) mod m
4. **if** T[j] = -1 **then** T[j]=k **else** i=i+1
5. **if** i=m **then output** << „Zu viele Schlüssel in der Hash-Tabelle“

Datenstrukturen

Hash-Suche(T,k**)**

1. i=0
2. **while** i<m **and** T[j]≠-1 **do**
3. j=(h(k)+i) mod m
4. **if** T[j] = k **then return** j
5. i=i+1
6. **return** -1

Datenstrukturen

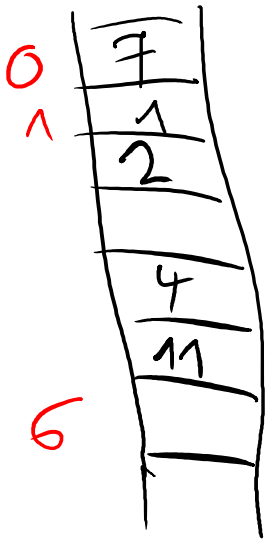
Aufgabe

- Fügen Sie die Schlüssel 2,4,7,1,11 in eine Hash-Tabelle mit offener Adressierung mit linearem Ausprobieren ein
- Die Hash-Funktion ist h(k) = k mod 7
- Die Tabelle hat Größe 7

Datenstrukturen

Aufgabe

- Fügen Sie die Schlüssel 2,4,7,1,11 in eine Hash-Tabelle mit offener Adressierung mit linearem Ausprobieren ein
- Die Hash-Funktion ist h(k) = k mod 7
- Die Tabelle hat Größe 7



Datenstrukturen

Löschen in Hash-Tabellen mit Linearem Ausprobieren

- Schwierig: Wir können nicht einfach einen gelöschten Schlüssel mit -1 markieren
- Eine Lösung: Markiere gelöschte Elemente mit DELETED (bzw. -2)
- Einfügen kann dann in Stellen schreiben, in denen DELETED steht
- Suche ist wie bisher (läuft weiter, wenn eine Zelle mit DELETED gefunden wird)

Datenstrukturen

Hash-Tabellen

- Hash-Funktion bildet Universum auf kleine Hash-Tabelle ab
- Kollisionen können mit Verkettung aufgelöst werden
- Verschiedene Möglichkeiten, Hash-Funktion zu wählen
- Offene Adressierung vermeidet Zeiger

Hier keine Analyse

- Universelles Hashing: Durchschnittliche (erwartete) Laufzeit für Suchen, Einfügen und Löschen ist O(1+n/m), wobei n die Anzahl gespeicherter Schlüssel und m die Größe der Hash-Tabelle ist

Datenstrukturen

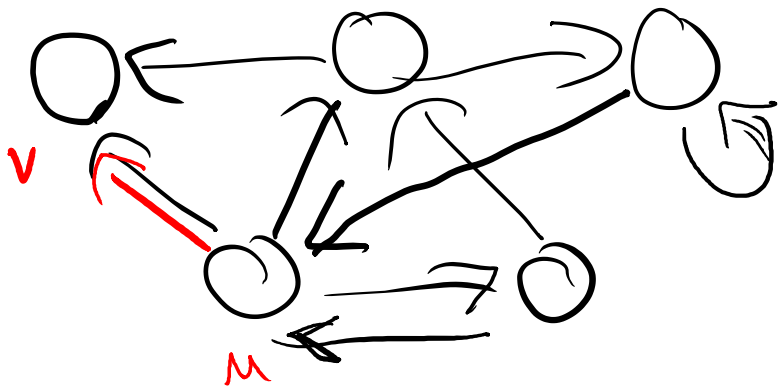
Zusammenfassung

- Elementare Datenstrukturen
  - Feld
  - Sortiertes Feld
  - Liste
- Binäre Suchbäume
- Rot-Schwarz Bäume
- Hashing

Graphalgorithmen

Definition (gerichteter Graph)

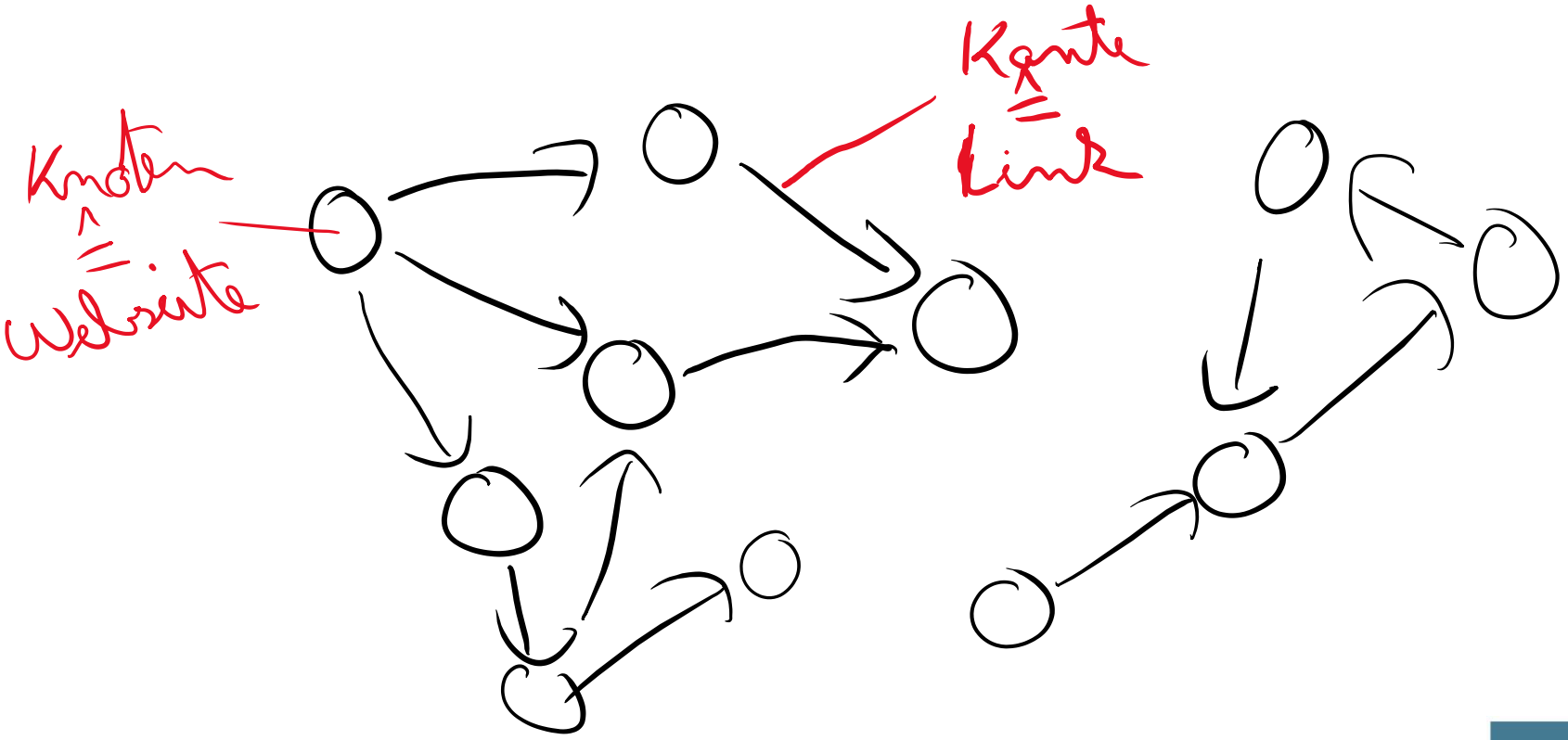
- Ein *gerichteter Graph* ist ein Paar (V,E), wobei V eine endliche Menge ist und $E \subseteq V \times V$.
- V heißt Knotenmenge des Graphen
- Die Elemente aus V sind die Knoten des Graphen
- E heißt Kantenmenge des Graphen
- Die Elemente aus E sind die Kanten des Graphen



$(v,w) \in E$

Graphalgorithmen

Beispiel: Repräsentation des Webgraph



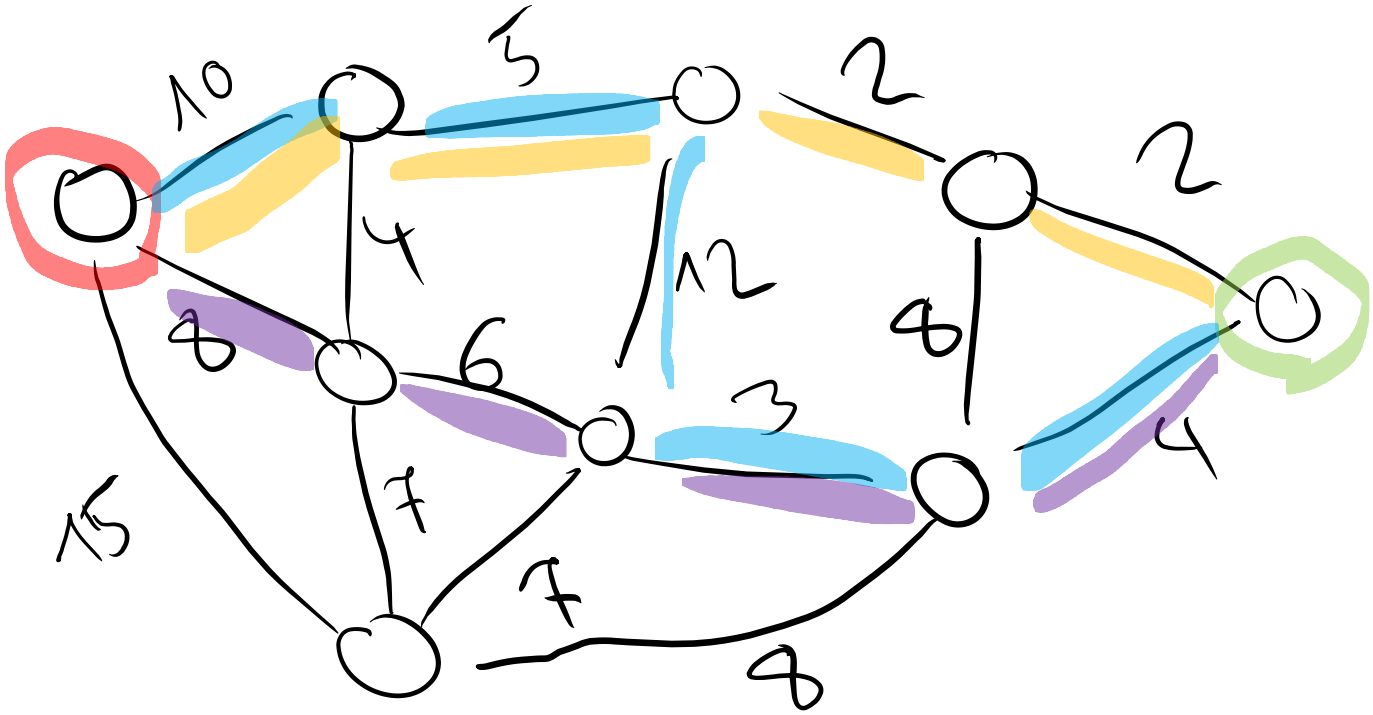
Graphalgorithmen

Definition (ungerichteter Graph)

- Ein *ungerichteter Graph* ist ein Paar (V,E), wobei V eine endliche Menge ist und E Teilmenge der Menge aller Paare von Elementen aus V ist
- V heißt Knotenmenge des Graphen
- Die Elemente aus V sind die Knoten des Graphen
- E heißt Kantenmenge des Graphen
- Die Elemente aus E sind die Kanten des Graphen
- Wir stellen Kanten aus V wie im gerichteten Fall durch (u,v) dar und nehmen an, dass die Kante (u,v) gleich der Kante (v,u) ist
- Manchmal repräsentieren wir einen ungerichteten Graph durch einen gerichteten, indem wir jede Kante (u,v) durch die gerichteten Kanten (u,v) und (v,u) ersetzen

Graphalgorithmen

Beispiel: Kürzeste Strecke zwischen zwei Orten



19

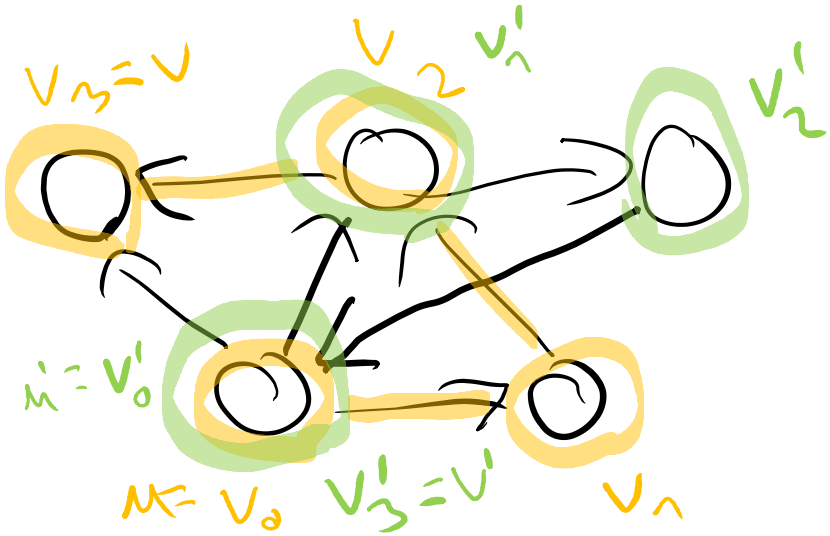
34

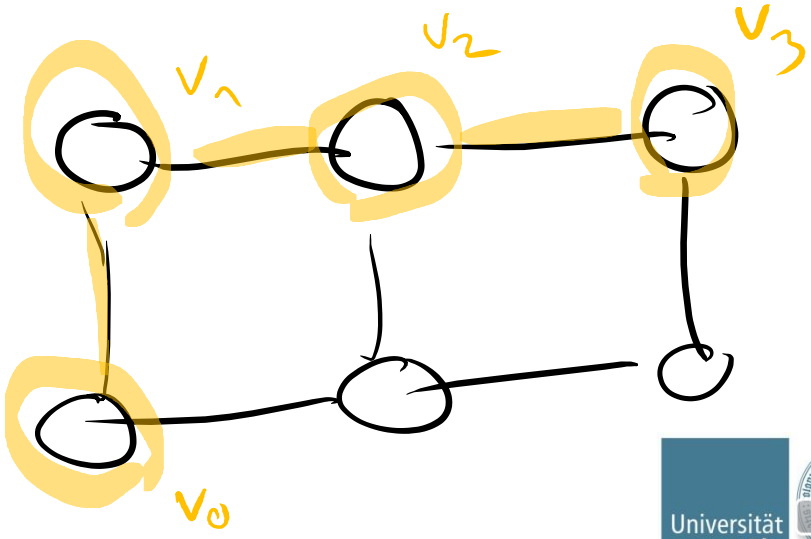
21

Graphalgorithmen

Definition (Weg)

- Ein *Weg* der Länge k von Knoten u zu Knoten v in einem Graph G=(V,E) ist eine Sequenz von k+1 Knoten $(v_0,\ldots, v_k)$ mit $u=v_0$ und $v = v_k$ und $(v_{i-1}, v_i)\in E$ für i=1,..,k.
- Wir sagen, dass v von u *erreichbar* ist, wenn es einen Weg von v nach u gibt
- Ein Weg heißt *einfach*, wenn kein Knoten auf dem Weg mehrfach vorkommt

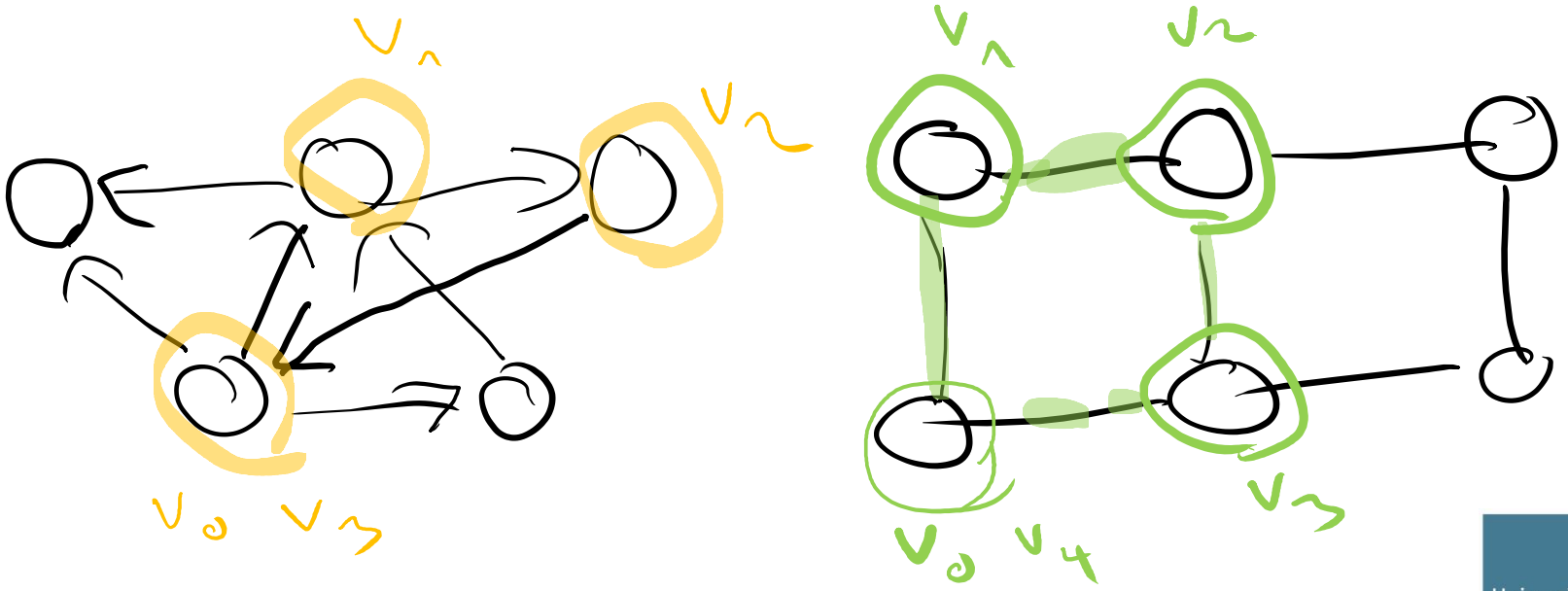




Graphalgorithmen

Definition (Kreis)

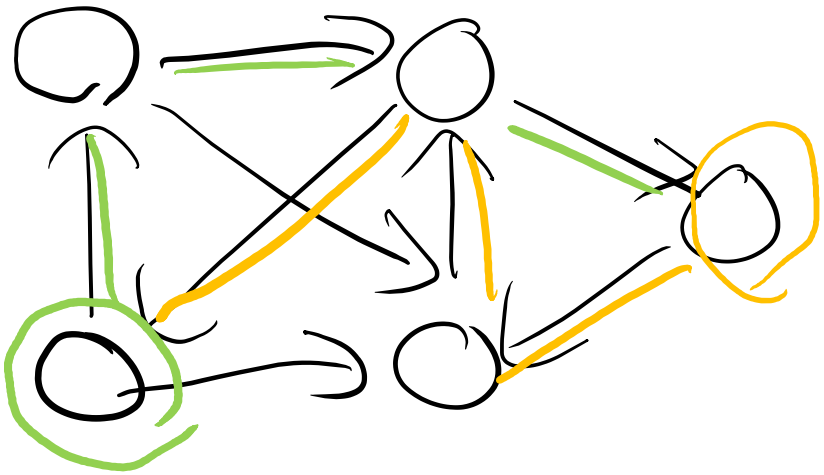
- Ein Weg $(v_0,\ldots, v_k)$ in einem ungerichteten (gerichteten) Graph heißt *Kreis*, falls $v_0=v_k$
- Ein Kreis $(v_0,\ldots, v_k)$ heißt *einfach*, wenn $(v_0,\ldots, v_{k-1})$ ein einfacher Weg ist

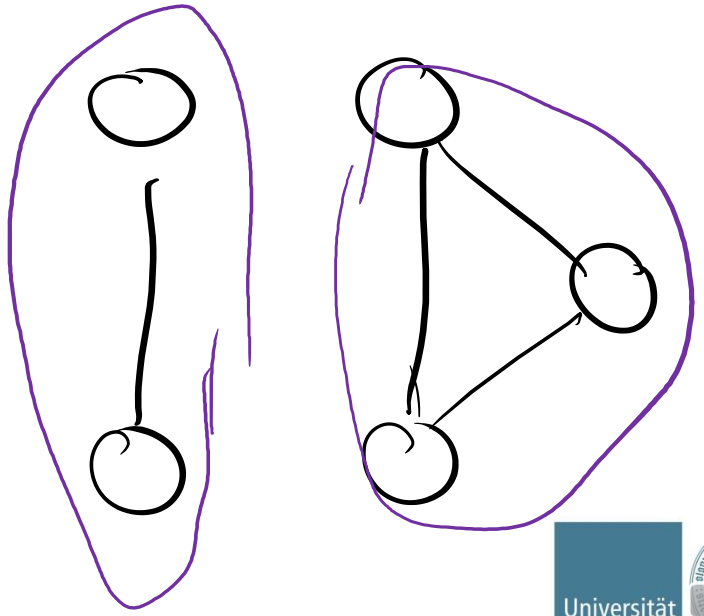


Graphalgorithmen

Definition (Zusammenhang)

- Ein gerichteter Graph heißt *stark zusammenhängend*, wenn es von jedem Knoten einen Weg zu jedem anderen Knoten im Graph gibt
- Ein ungerichteter Graph heißt *zusammenhängend*, wenn es von jedem Knoten einen Weg zu jedem anderen Knoten im Graph gibt

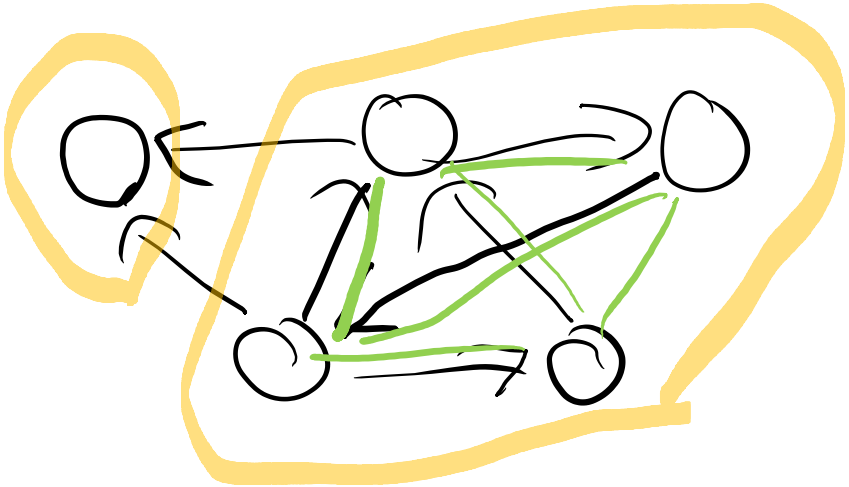


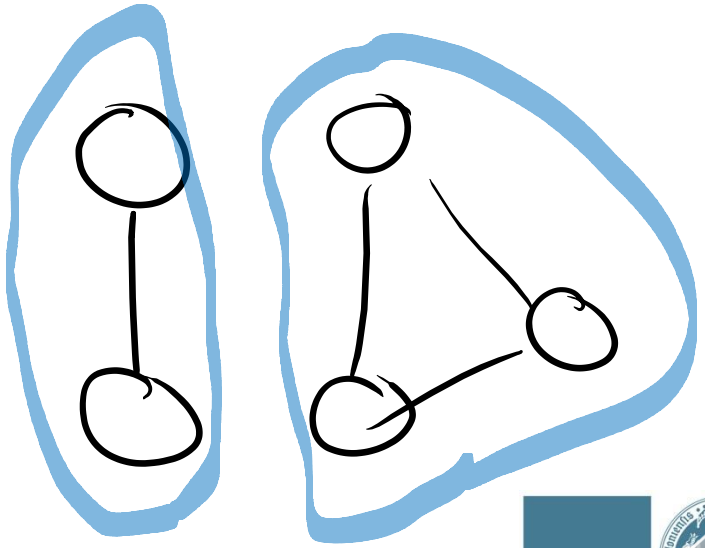


Graphalgorithmen

Definition (Zusammenhangskomponenten)

- Die *starken Zusammenhangskomponenten* eines Graphen sind die Äquivalenzklassen der Relation „ist beidseitig erreichbar“
- Die *Zusammenhangskomponenten* eines Graphen sind die Äquivalenzklassen der Relation „ist erreichbar“

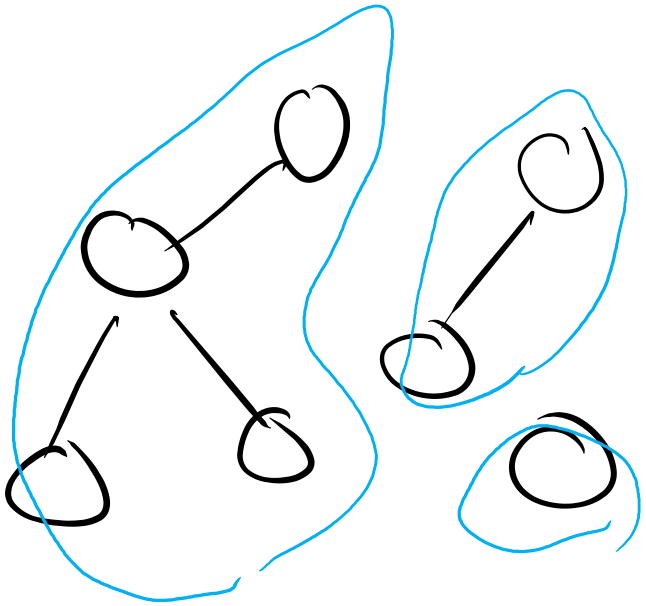


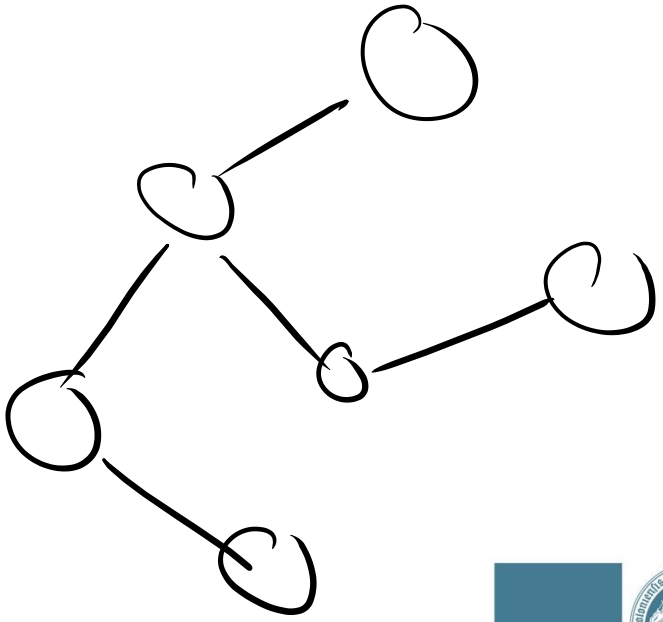


Graphalgorithmen

Definition (Baum und Wald)

- Ein kreisfreier ungerichteter Graph heißt *Wald*
- Ein ungerichteter, zusammenhängender, kreisfreier Graph heißt *Baum*

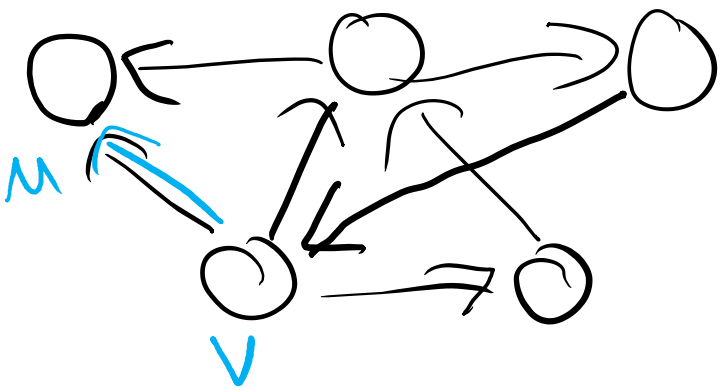


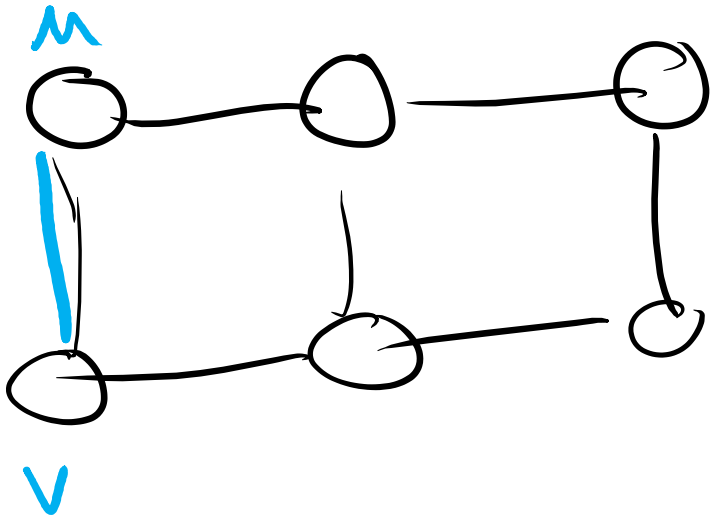


Graphalgorithmen

Definition (Nachbar)

- Ein Knoten u ist Nachbar eines Knotens v in einem gerichteten (ungerichteten) Graph G=(V,E), wenn es eine Kante (v,u)$\in$E gibt

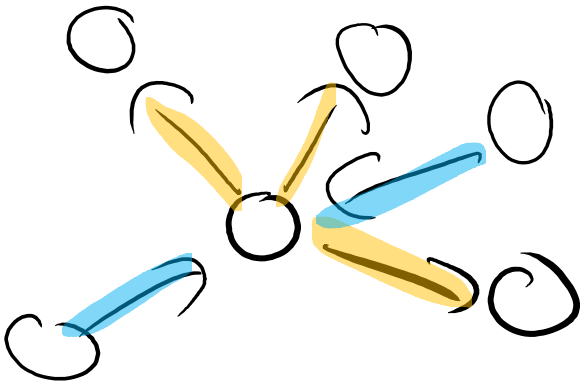


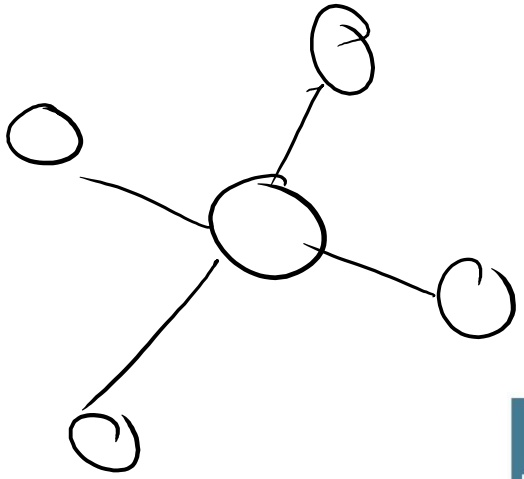


Graphalgorithmen

Definition (Knotengrad)

- Der *Ausgangsgrad* eines Knotens in einem gerichteten Graph ist die Anzahl Kanten, die den Knoten verlassen
- Der *Eingangsgrad* eines Knotens in einem gerichteten Graph ist die Anzahl Kanten, die auf den Knoten zeigen
- Der *Grad* eines Knotens v in einem ungerichteten Graph ist die Anzahl Kanten die an v anliegen

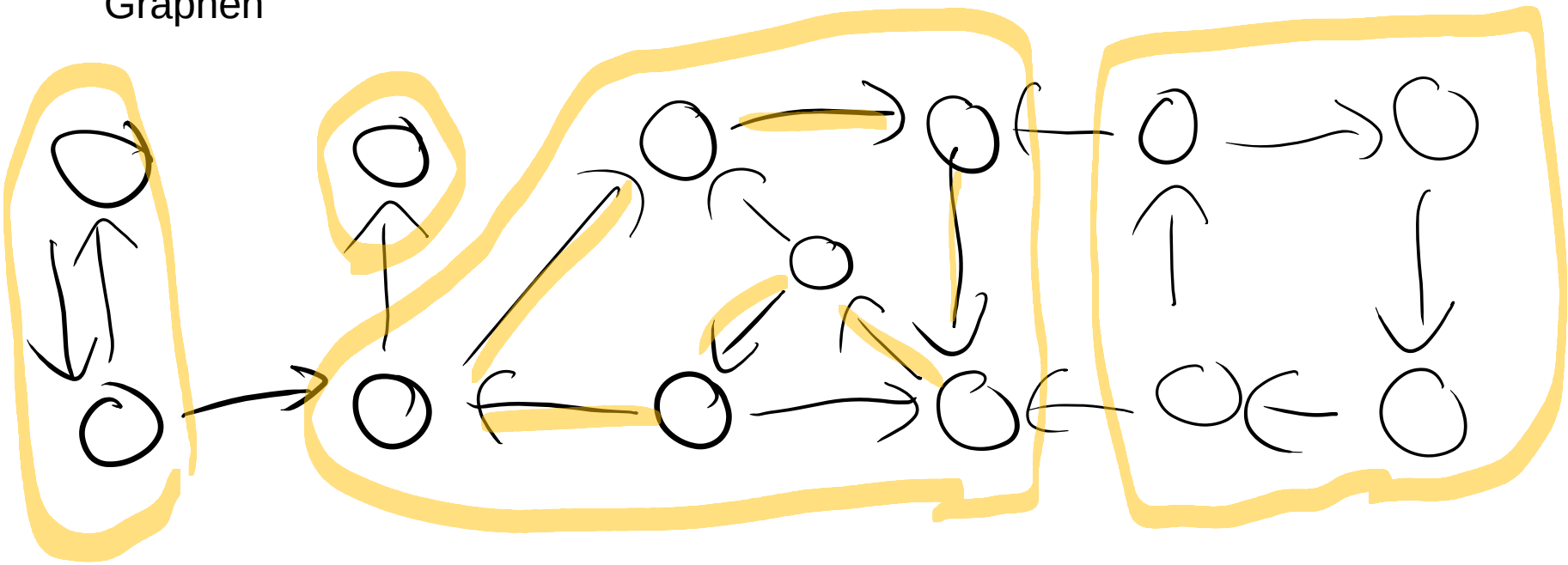




Graphalgorithmen

Aufgabe

- Bestimmen Sie die starken Zusammenhangskomponenten des folgenden Graphen



Graphalgorithmen

Datenstrukturen zur Repräsentation eines Graphen

- Adjazenzlisten: Dünn besetzen Graphen ($|E| << n^2$)
- Adjazenzmatrix: Dicht besetzte Graphen ($|E|$ nah an n^2)

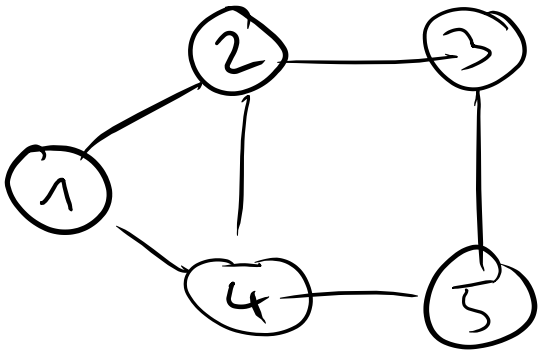
Arten von Graphen

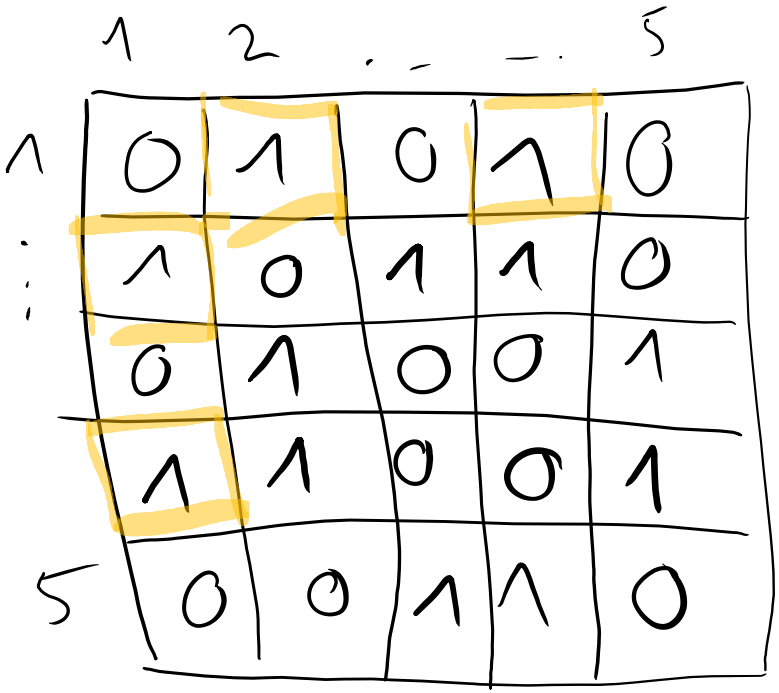
- Ungerichtet, gerichtet
- Ungewichtet, gewichtet (Knoten und/oder Kanten haben Gewichte)

Graphalgorithmen

Adjazenzmatrixdarstellung

- Knoten sind nummeriert von 1 bis |V|
- $|V| \times |V|$ Matrix $A = (a_{ij})$ mit
- $a_{ij} = 1$, wenn $(i,j) \in E$ und $a_{ij} = 0$, sonst
- Bei ungerichteten Graphen gilt $A = A^T$

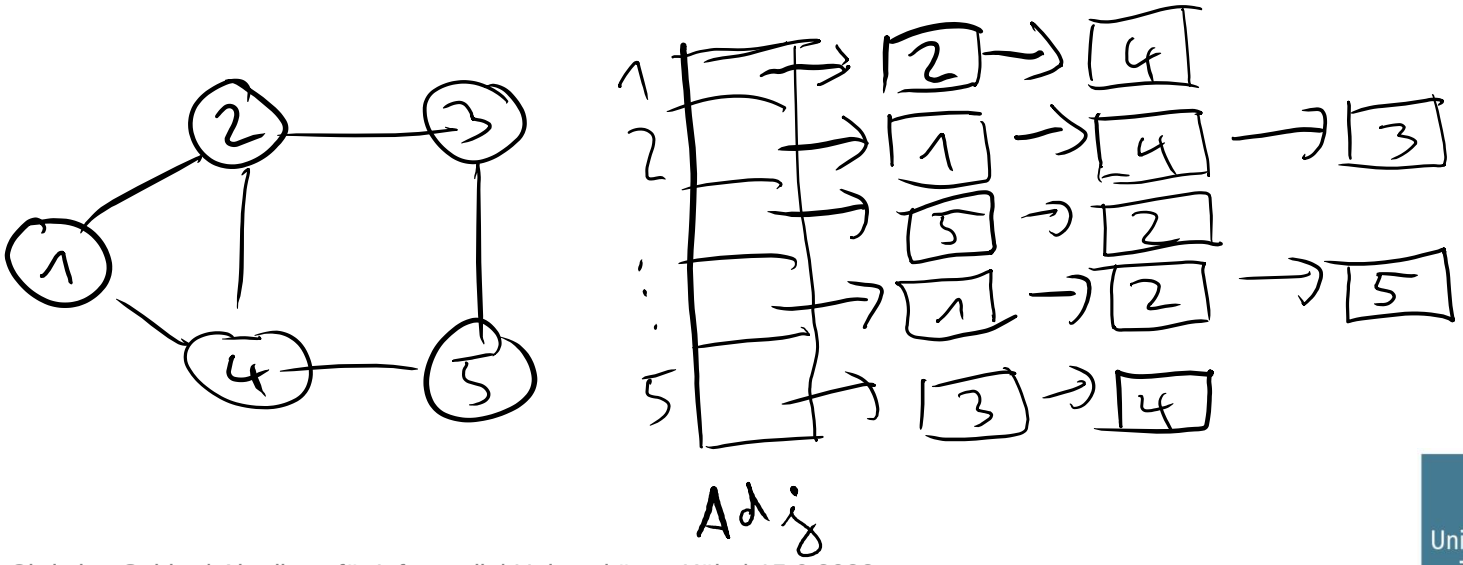




Graphenalgorithmen

Adjazenzlistendarstellung

- Feld Adj mit |V| Listen (eine pro Knoten)
- Für Knoten v enthält Adj[v] eine Liste aller Knoten u mit $(v,u) \in E$
- Die Knoten in Adj[v] heißen zu v benachbart
- Ist G ungerichtet, so gilt: $v \in Adj[u] \Leftrightarrow u \in Adj[v]$



Graphenalgorithmen

Graphen mit Kantengewichten

- Adjazenzmatrix: Gewicht einer Kante steht in der Adjazenzmatrix
- Adjazenzlisten: Gewicht w(u,v) von Kante (u,v) wird mit Knoten v in u‘s Adjazenzliste gespeichert

Zusammenfassung

- Direkte Adressierung
- Hash-Tabellen
  - Auflösen von Kollisionen
  - Wahl der Hash-Funktion
  - Offene Adressierung
- Graphalgorithmen
  - Grundlegende Begriffe der Graphentheorie
  - Datenstrukturen zum Speichern von Graphen

Referenzen

- T. Cormen, C. Leisserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT press. Second edition, 2001.