

Algorithmen im Chip-Entwurf 1

Probleme, Werkzeuge und Graphen

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

Orga 1 - Material

- **Grundlage der Vorlesung**
 - *Algorithms for VLSI Design Automation*
Sabih H. Gerez
 - In Informatikbibliothek vorhanden
- **Wissenschaftliche Arbeiten („Papers“)**
- **Wissenstiefe**
 - Kein perfektes Verständnis ...
 - ... aber Überblick über das Material
 - ◆ Fragen stellen!

Probleme, Werkzeuge und Graphen

Orga 2 - Prüfungsmodus

- **Idealerweise in Projektarbeit: 4 SWS**
 - Programmierung, Kolloquien, Vorträge
 - Viel Arbeit: 15K-20K LoC in Java
 - Hierfür aber max. 18 Plätze (betreuungsintensiv!)
- **Für alle anderen: 2 SWS**
 - Normale vorlesungsbegleitende Prüfung
 - Zwei Teilprüfungen
 - Je nach Andrang mündlich oder schriftlich
- **Aufteilung: Freiwillig 2 SWS, oder ...**
 - Abgabe von Lösung der ersten Aufgabe
 - Beste Abgaben gehen in 4 SWS-Modus ein

Probleme, Werkzeuge und Graphen

Orga 3 - Teilklausuren

- Nur im Prüfungsmodus 2 SWS relevant
- Geplant in KW 48 und KW 6
- Je 45 Minuten, erreichbar je 45 Punkte
 - Werden für Endnote addiert
- Einbringen der ersten Programmieraufgabe
 - War ja für alle
 - Mit maximal 10 Punkten, wird aufaddiert
 - Kann Note um bis zu einen Wert (1,0) anheben
 - Zum Bestehen aber mindestens 27 „echte“ Klausurpunkte erforderlich

Orga 4 – Benotung 4 SWS

- Viel Freiheit bei der Realisierung
- Keine starren Bewertungsrichtlinien
 - Analog zu Diplom-Arbeit etc.
- Grundideen
 - Brauchbar kommentierte, brauchbar dokumentierte und funktionierende Lösung der Aufgabenstellung: 2,0
 - ◆ Kleinere Schwächen: OK
 - ◆ Einbußen in Lösungsqualität, Rechenzeit, Speicher, ...
- Aber Luft nach oben (Richtung 1,0), z.B. für
 - ◆ Sehr gute eigene Algorithmen und Datenstrukturen
 - ◆ Umfassende Kommentierung und Dokumentation
 - ◆ Sehr gute Lösungsqualität
 - ◆ Kurze Rechenzeiten
 - ◆ Niedriger Speicherverbrauch

Probleme, Werkzeuge und Graphen

Orga 5 - Prüfungsleistung

- **Benotete Prüfungsleistung**
- **Beginnend in 4. Semesterwoche (1. Abgabe)**
- **Gewertet**
 - ◆ **Programme**
 - ◆ Funktion, Code-Qualität, (Dokumentation)
 - ◆ **Kolloquien**
 - ◆ **Vorträge**
- **Individuelle Prüfung**
- **Nur in Zweifelsfällen**
- **Bei nicht nachvollziehbarer Mitarbeit**

Orga 6 - Aufbau

- **Integrierte Veranstaltung**
 - **Zu Beginn: Nur Vorlesung (2 pro Woche)**
 - **Dann: praktische Programmierarbeit**
 - ◆ **In Gruppen**
 - ◆ **Kolloquien**
 - ◆ **Vorträge**
 - **Vorlesung nun 1/Woche, am Ende keine mehr**
- **Kick-Off zu den praktischen Arbeiten**
 - **Anfang KW 43 (= 2. Semesterwoche)**
 - **Vorher Leitfaden lesen!**

Orga 6 – Zeitplan und WWW

- **Zeitplan**
 - **Vorlesung**
 - ◆ KW 42&43: Di+Fr, KW 44...4: Nur Di 11:40-13:20
 - ◆ Keine mehr in KW 5&6
 - **Projektarbeit**
 - ◆ Abgaben KW 45, 49, 3, 5: Mo 23:59
 - ◆ Vorträge KW 45, 49, 3, 6: Fr 9:50-11:30
 - ◆ Kolloquien KW 45, 49, 3, 5: Do nachmittags
- **Web-Seite**
 - <http://www.esa.informatik.tu-darmstadt.de>
Unterpunkt „Lehre“
 - **Material und Ankündigungen**

Überblick

- **VLSI Entwurf**
 - Probleme
 - Bereiche
 - Tätigkeiten→ **Werkzeuge**
- **Hierarchie und Abstraktion**
- **Algorithmische Graphentheorie**
 - Strukturen
 - Verfahren

VLSI Entwurfsproblem

„Implementiere eine Spezifikation in Hardware und optimiere dabei ...“

- Fläche (min.)
- Stromverbrauch (min.)
- Geschwindigkeit (max. oder passend)
- Entwurfszeit (min.)
- Testbarkeit (max.)

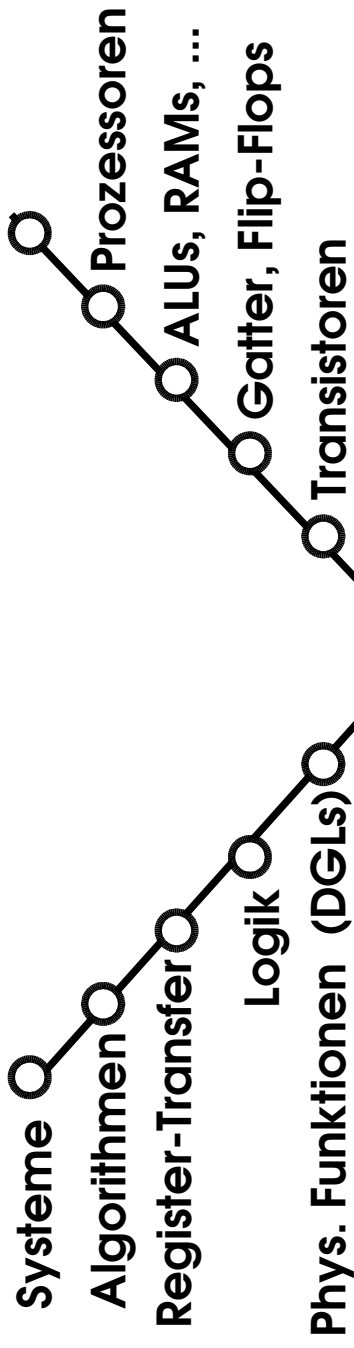
† „Alles auf einmal“ ist zu komplex

→ Aufteilen und vereinfachen

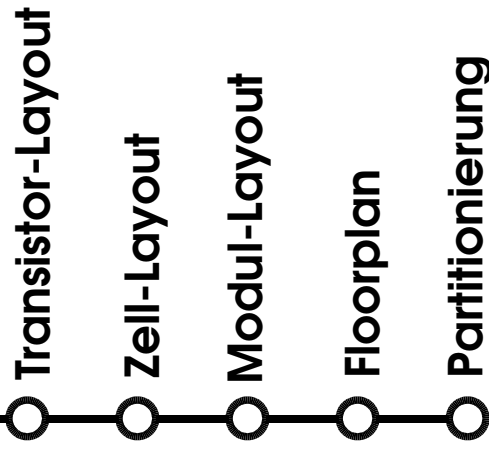
- Qualitätseinbussen

Entwurfsbereiche - Gajskis "Y"

Verhalten



Struktur



Physikalisch

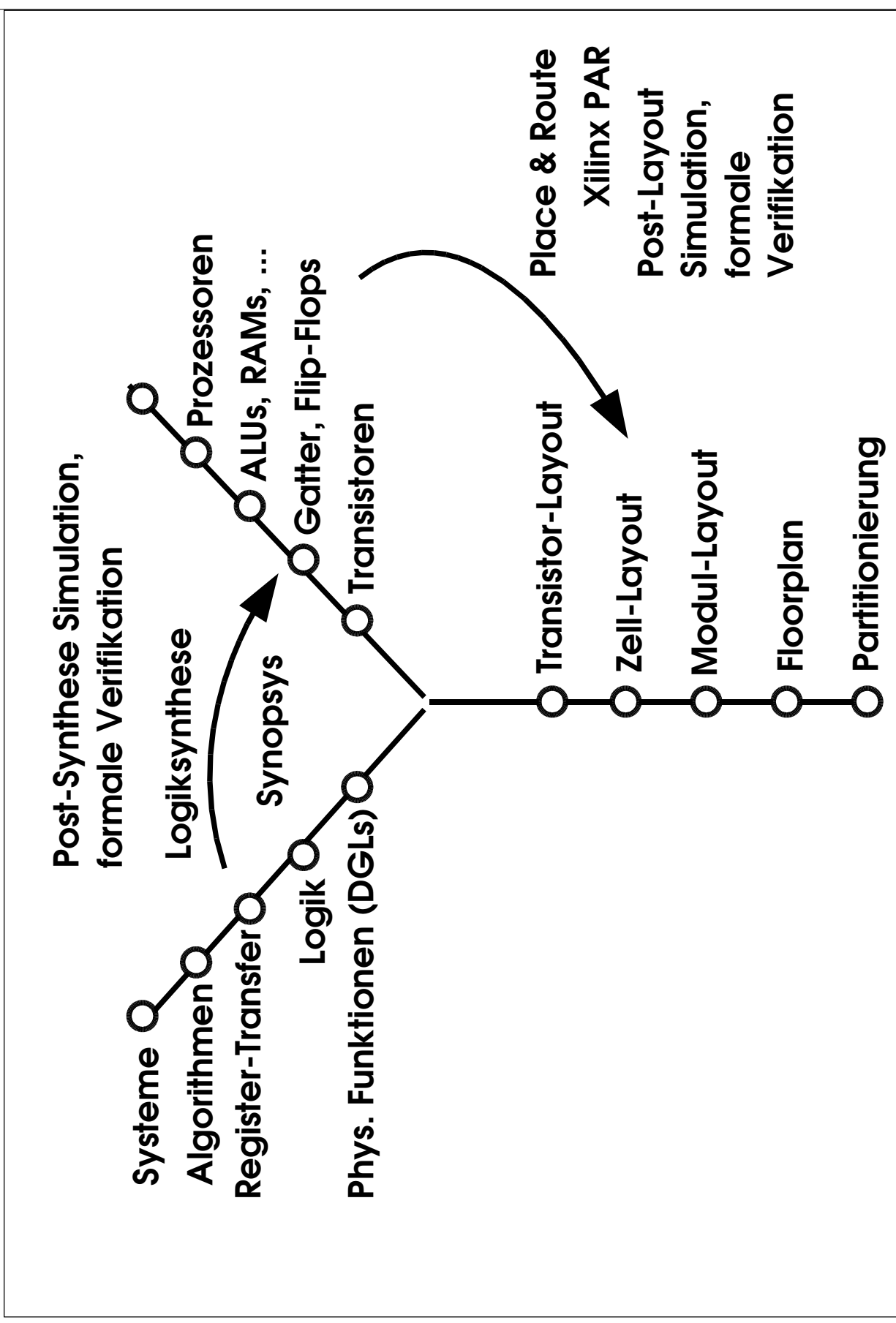
Probleme, Werkzeuge und Graphen

Tätigkeiten

- **Synthese**
 - Mehr Details durch Anwendung von Regeln
- **Verifikation**
 - Vergleiche Ergebnis mit Spezifikation
- **Analyse**
 - Untersuche Eigenschaften eines Ergebnisses
- **Optimierung**
 - Verbessere ein Ergebnis
- **Datenverwaltung**

Probleme, Werkzeuge und Graphen

Werkzeuge

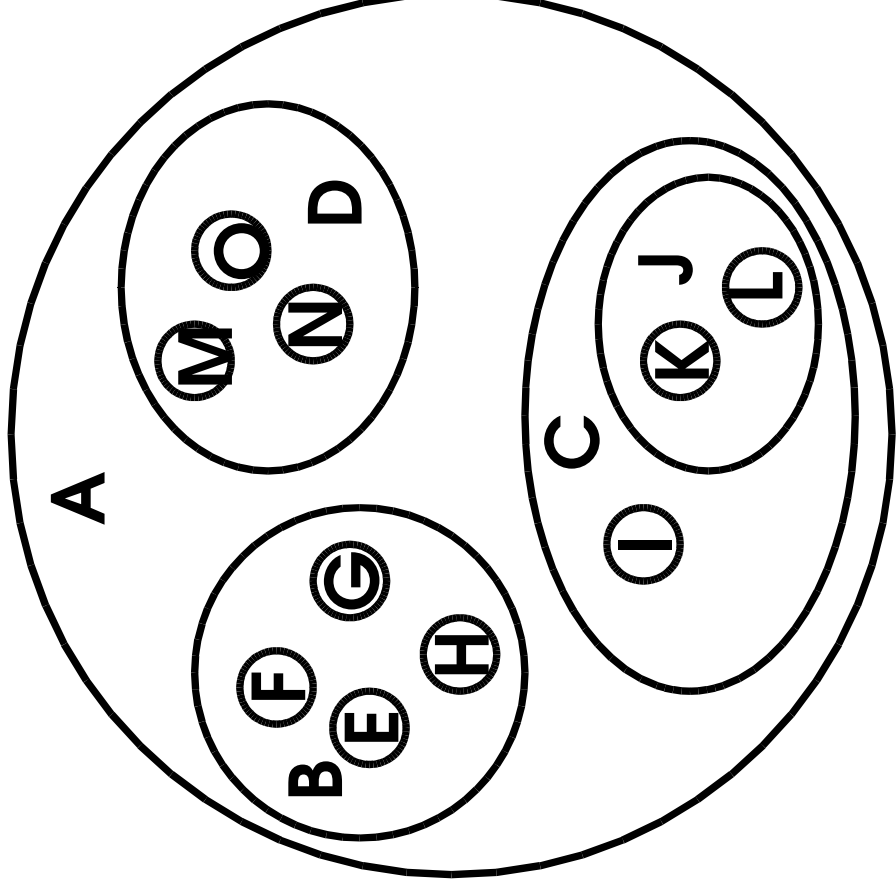
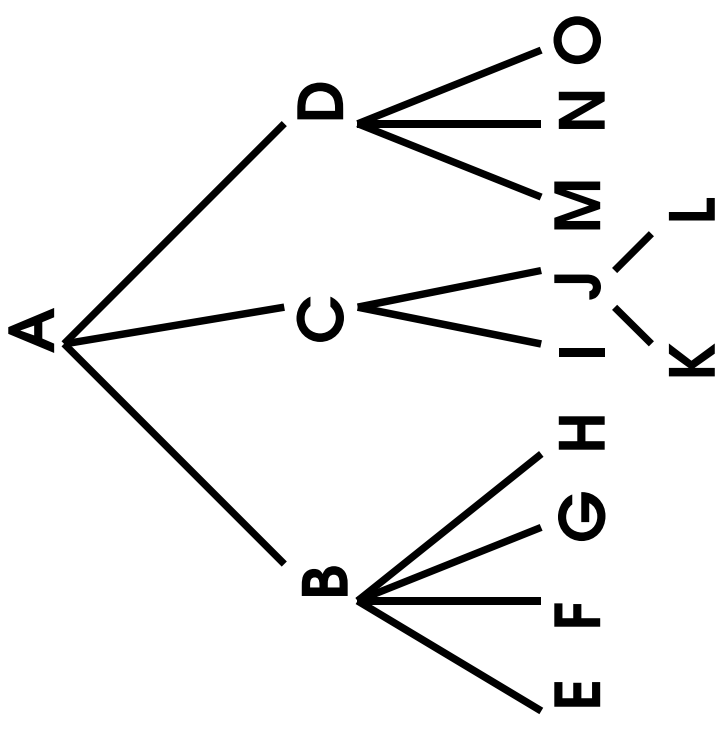


Probleme, Werkzeuge und Graphen

Strukturierung

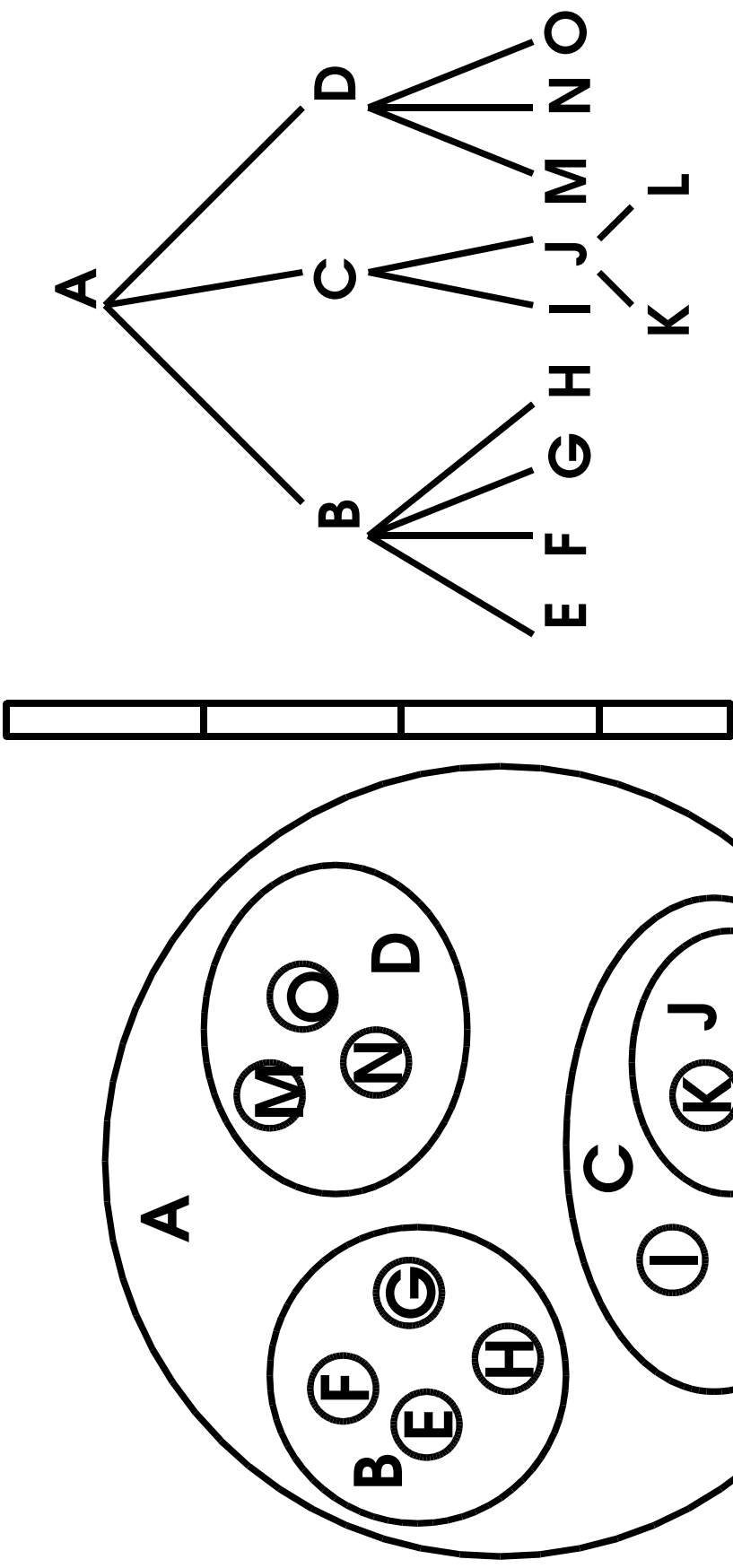
A
H I
B G J F
D C K
L

Hierarchie



Abstraktion

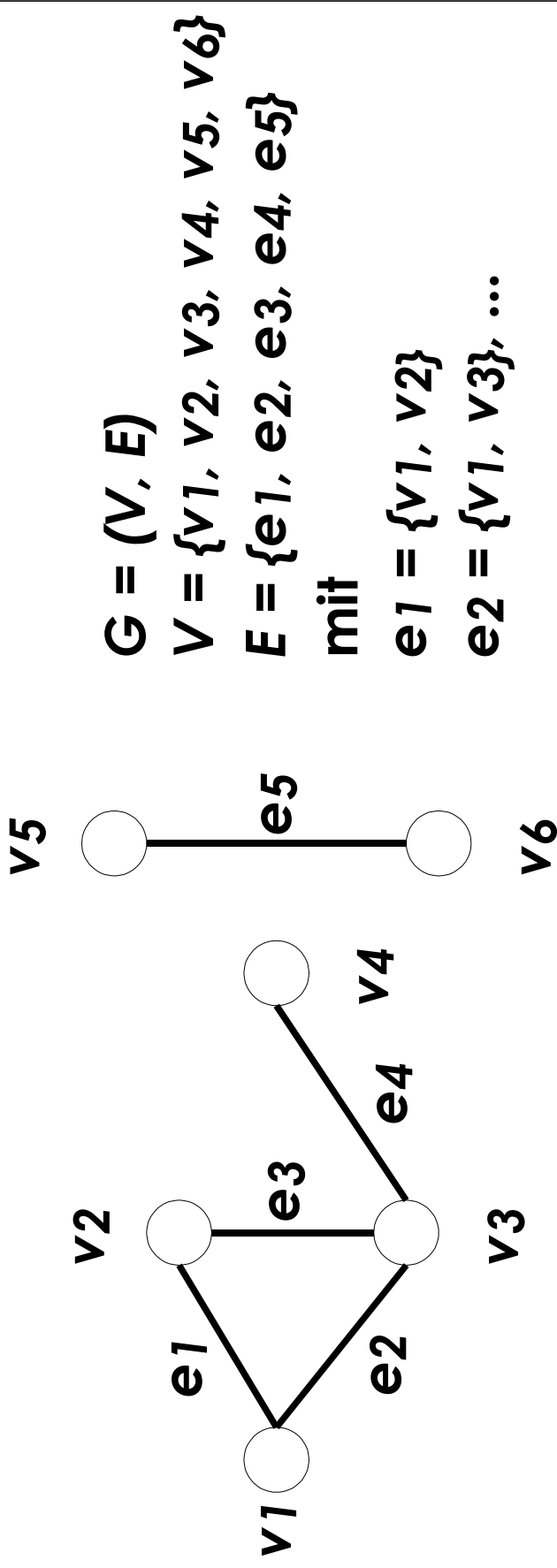
Abstraktionsebene



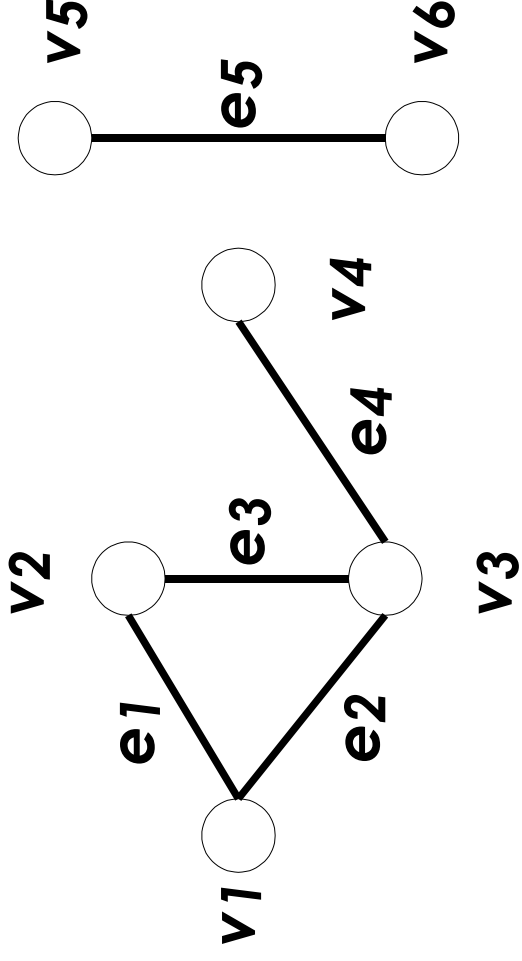
Probleme, Werkzeuge und Graphen

Graphentheorie

- Graph $G (V, E)$
 - Eine Menge V von Knoten (vertex)
 - Eine Menge E von Kanten (edge)
 - ◆ Kante $e = \{v_1, v_2\}$ verbindet Knoten v_1 und v_2

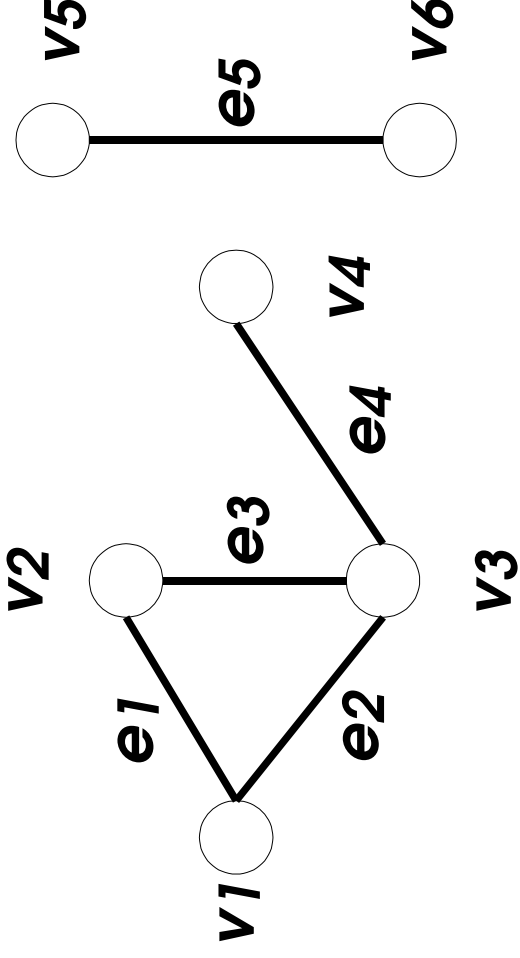


Inzidenz, Adjazenz und Grad



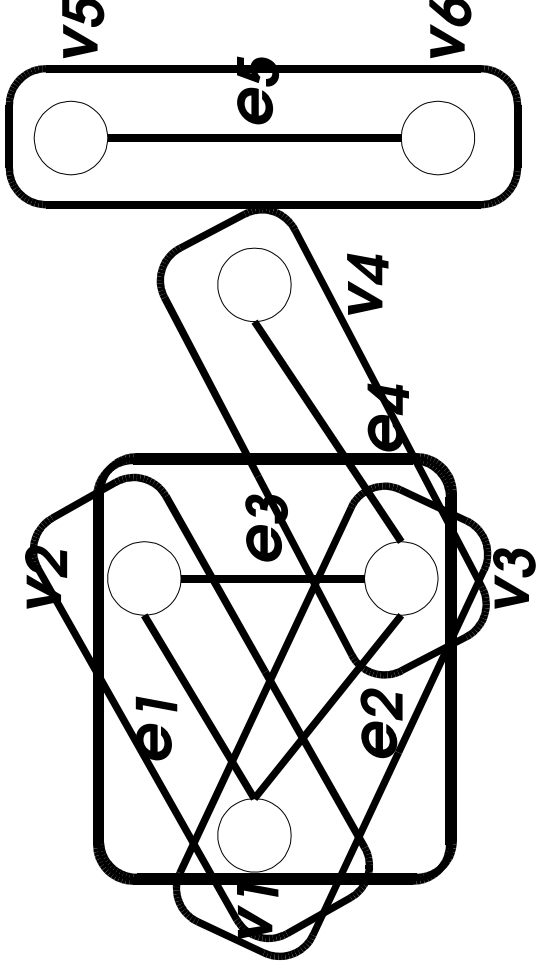
- $e = \{u, v\} \in E$
- e ist inzident u *incident*
- e ist inzident v *incident*
- u ist adjazent v *adjacent*
- $\text{Grad } g(v) = |\{e \in E \mid v \in e\}|$ *degree*

Subgraphen



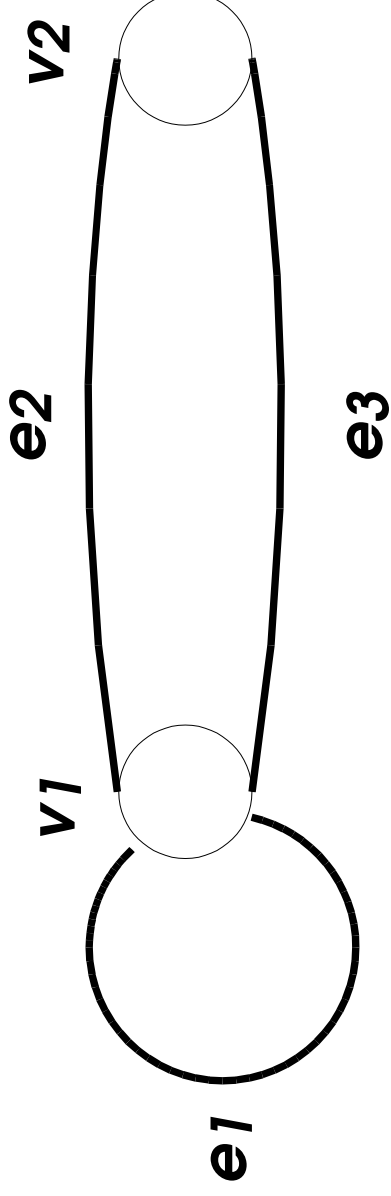
- Subgraph durch Entfernen von Knoten
- Entferne $v \in V$
- Entferne Kanten inzident zu v

Vollständigkeit und Cliques



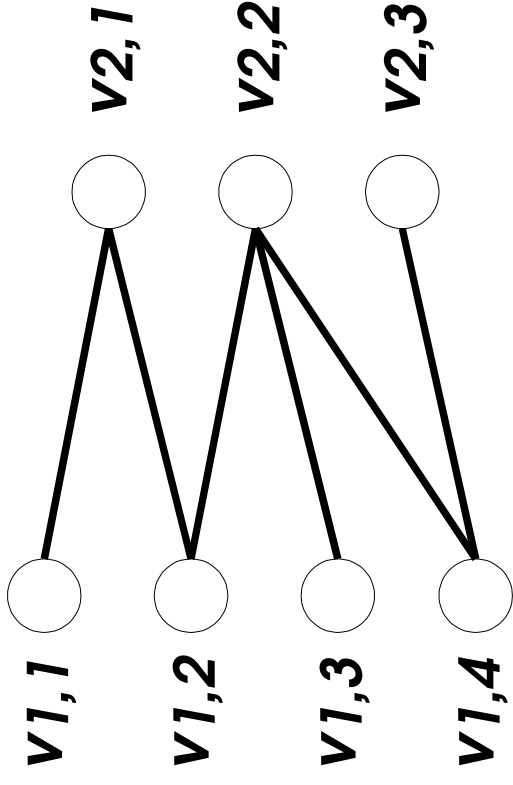
- **Komplett untereinander verbundene Knoten bilden vollständigen Graph (complete graph)**
- **Maximal ausgedehnte vollständige Graphen bilden Cliques**

Schlingen, parallele Kanten



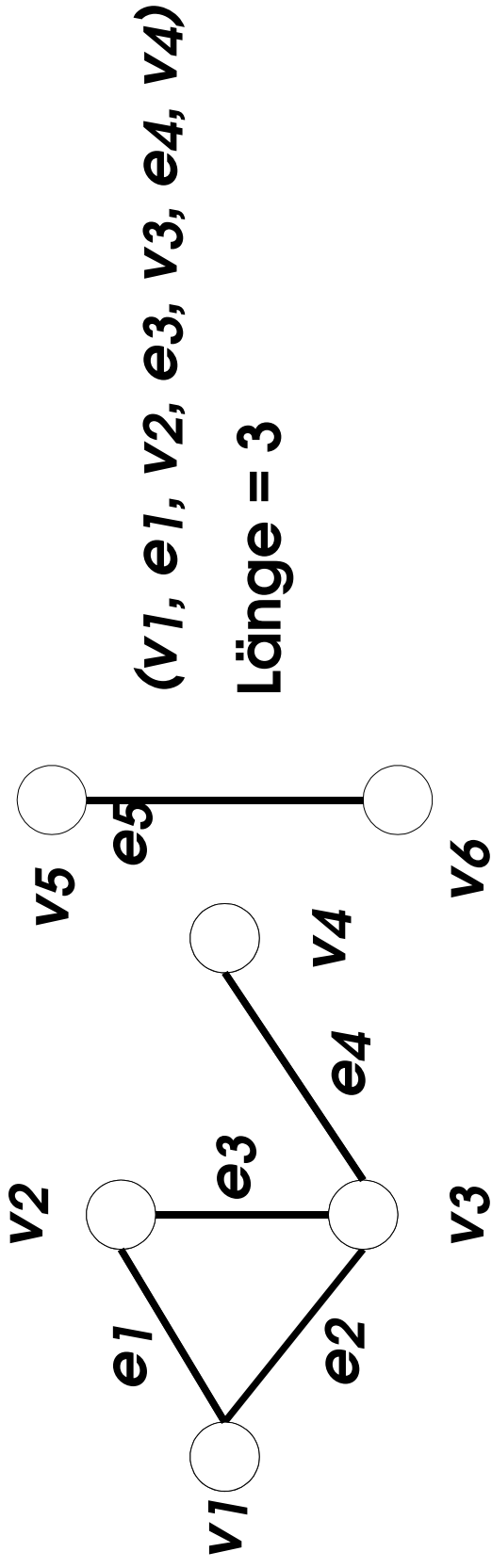
- e_1 Schlinge (selfloop)
- e_2, e_3 parallele Kanten
- einfache Graphen: weder noch (simple)
- Multigraphen: parallele Kanten OK

Bipartite Graphen



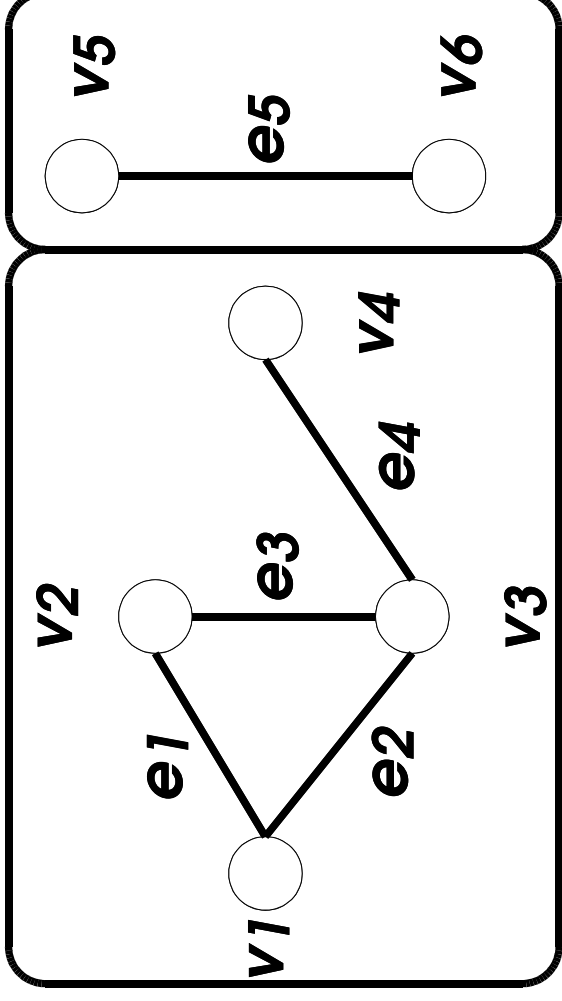
- Kanten nur zwischen Knoten aus nichtüberlappenden Mengen
- $G = (V1, V2, E)$ ist bipartiter Graph
 - $V1 \cap V2 = \emptyset$
 - $E = \{u, w\} \mid u \in V1 \wedge w \in V2\}$

Wege und Zyklen



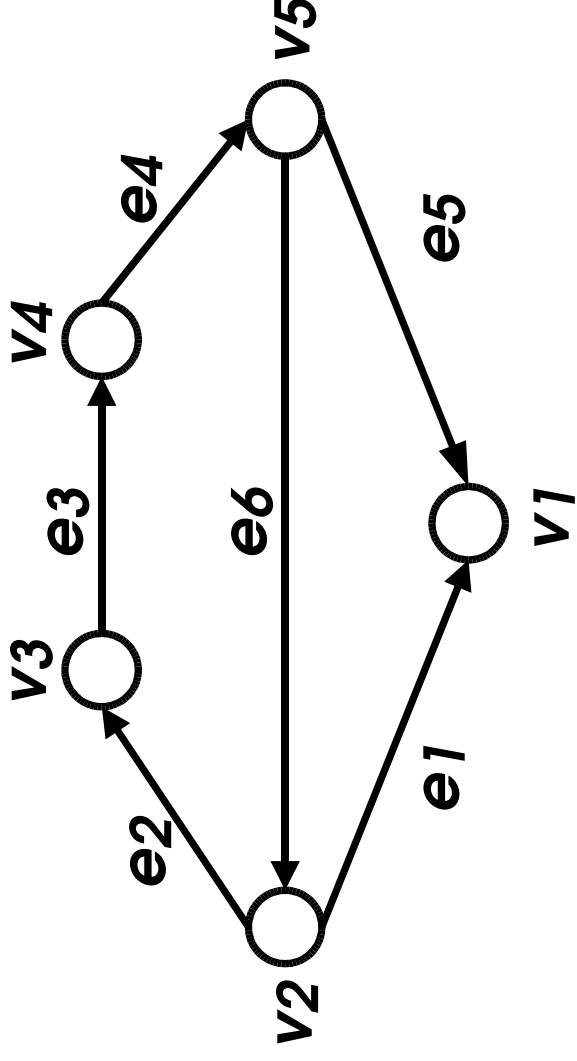
- **Weg: Folge von Knoten und Kanten**
 - **Beginnend und endend mit Knoten**
- **Länge: Anzahl der Kanten**
- **Zyklus: Anfang = Ende**

Zusammenhang



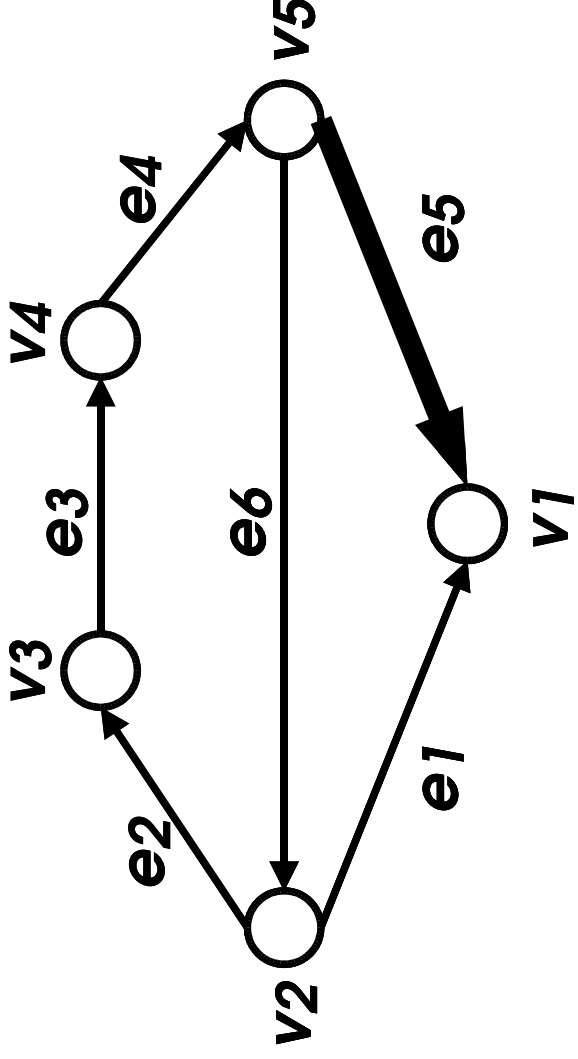
- u hängt mit v zusammen
- Es gibt einen beide verbindenden Weg
- Zusammenhängender Graph
- Alle Knoten hängen zusammen.
- Zusammenhängende Komponente
- Maximale zusammenhängende Subgraphen

Gerichtete Graphen



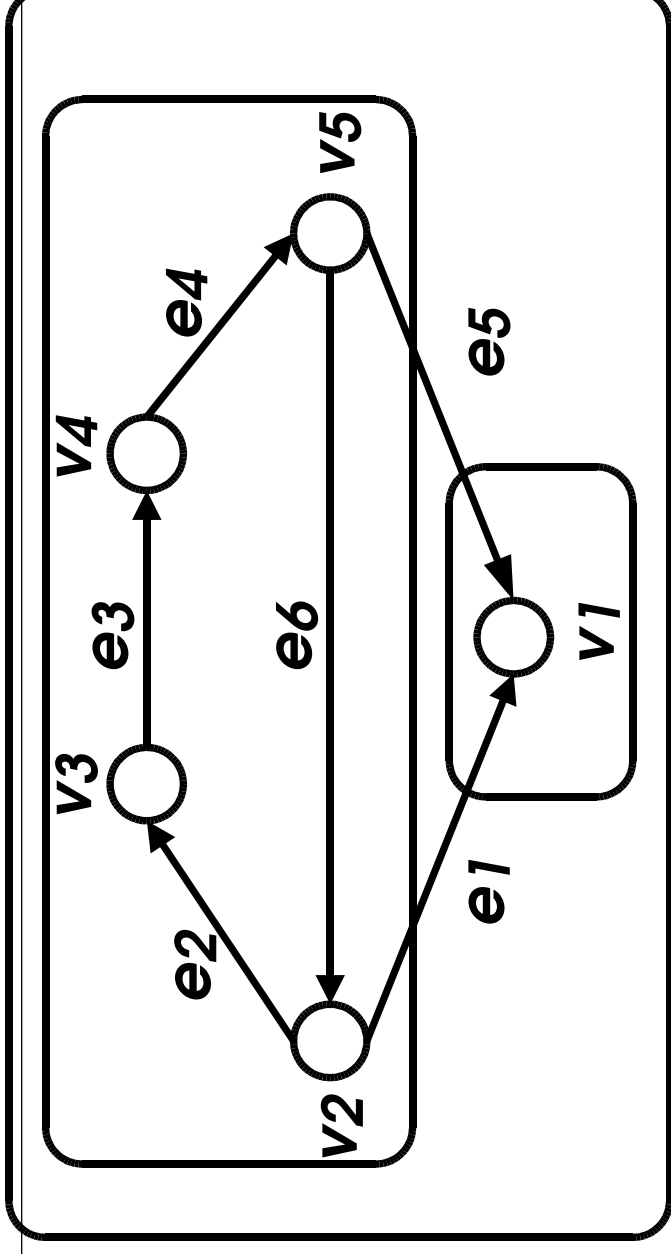
- $G(V, E)$ mit $e = (u, v) \wedge u, v \in E$
- e inzident von u (ausgehend)
- e inzident nach v (eingehend)
- Außengrad: Anzahl ausgehender Kanten
- Innengrad: Anzahl eingehender Kanten

Wege und Zyklen



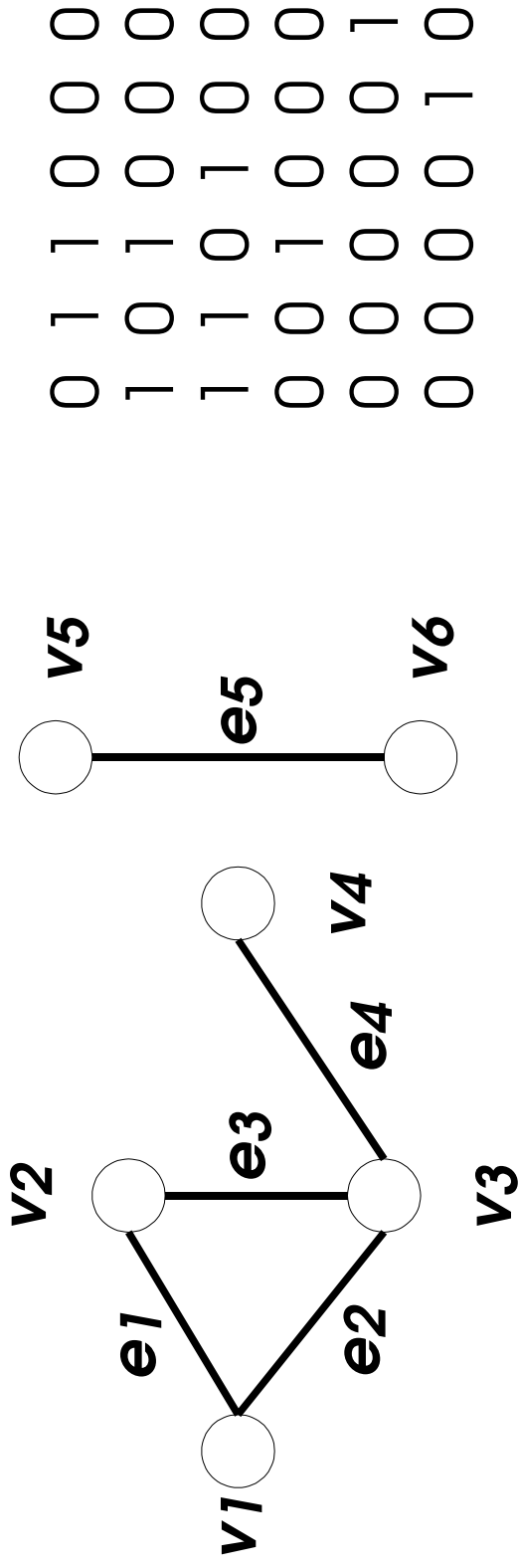
- Gerichteter Weg
- Gerichteter Zyklus
- Weg und Zyklus gelten auch noch!

Zusammenhang



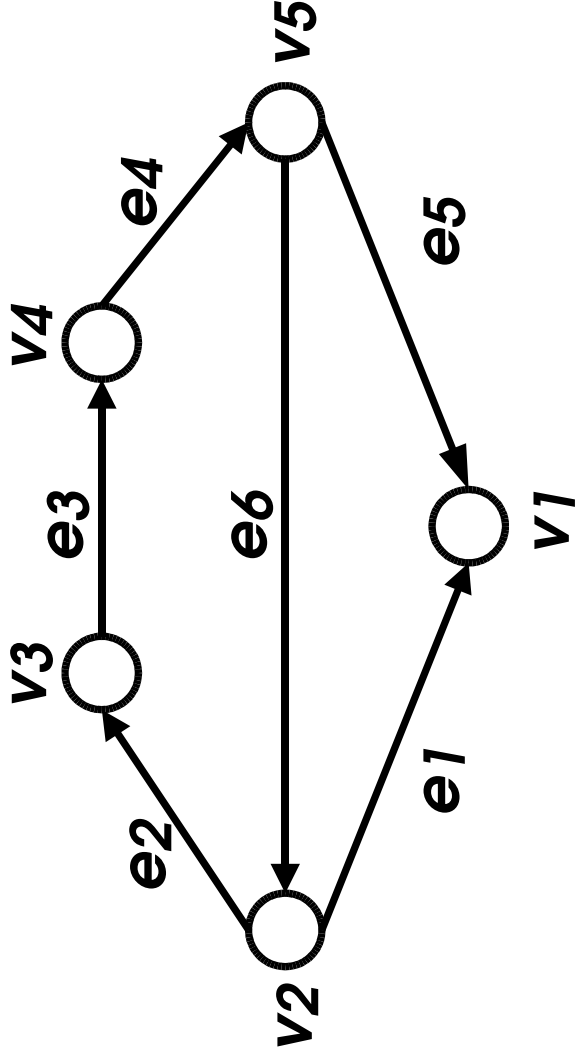
- **Starker Zusammenhang**
 - Gerichteter Weg von u nach v & von v nach u
- **Stark zusammenhängende Komponente**
 - Alle enthaltenen Knoten hängen stark zusam.
- **Schwacher Zusammenhang: Weg**

Datenstrukturen für Graphen



- Adjazenzmatrix AG von $G(V, E)$
 - $n \times n$ Matrix mit $n = |V|$
 - $A_{ij} = 1$ falls $\{v_i, v_j\} \in E$, sonst $= 0$
 - Symmetrische Matrix

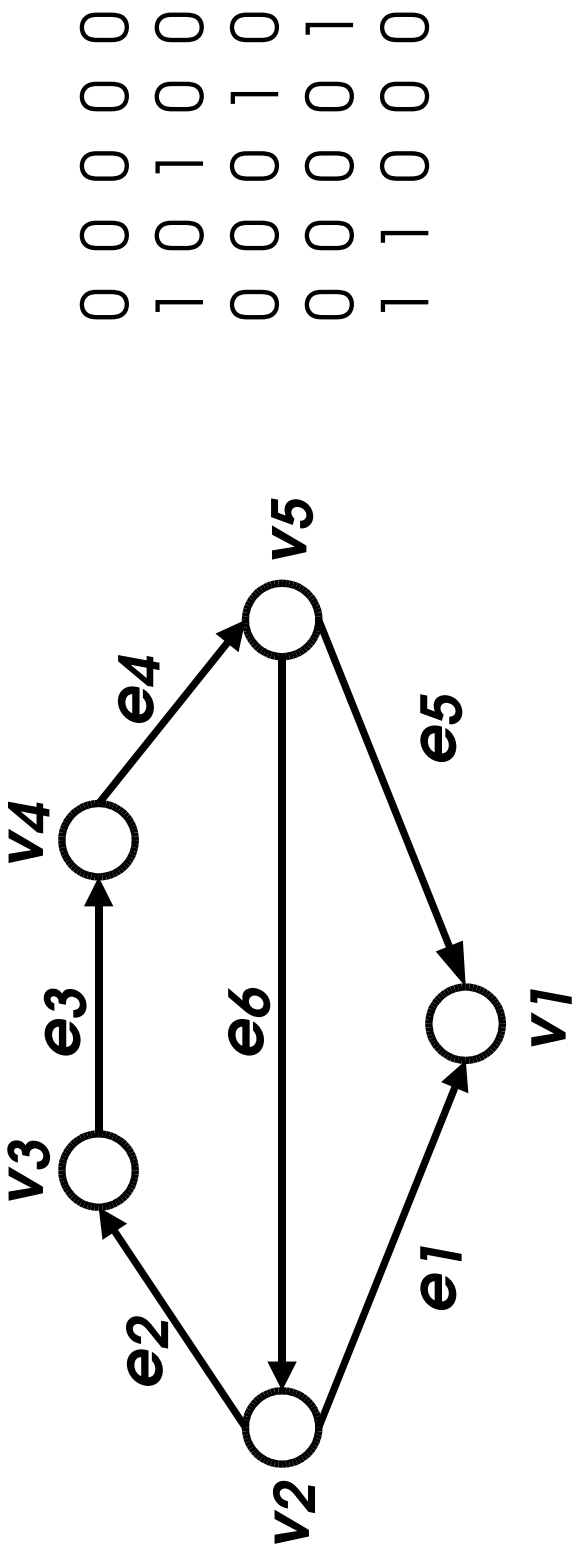
AG für gerichtete Graphen



0	0	0	0	0	0	0
1	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0
1	1	0	0	0	0	0

- Matrix nicht mehr symmetrisch

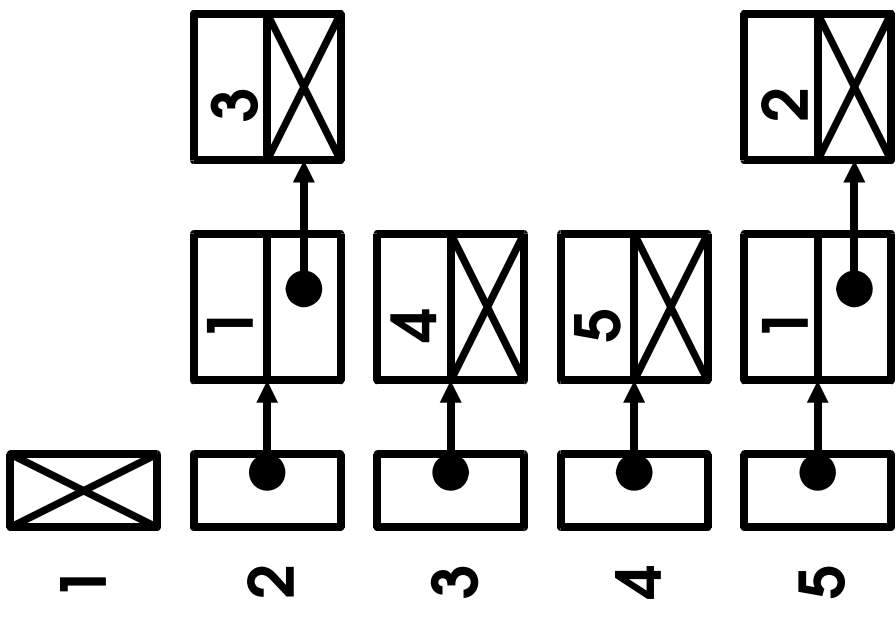
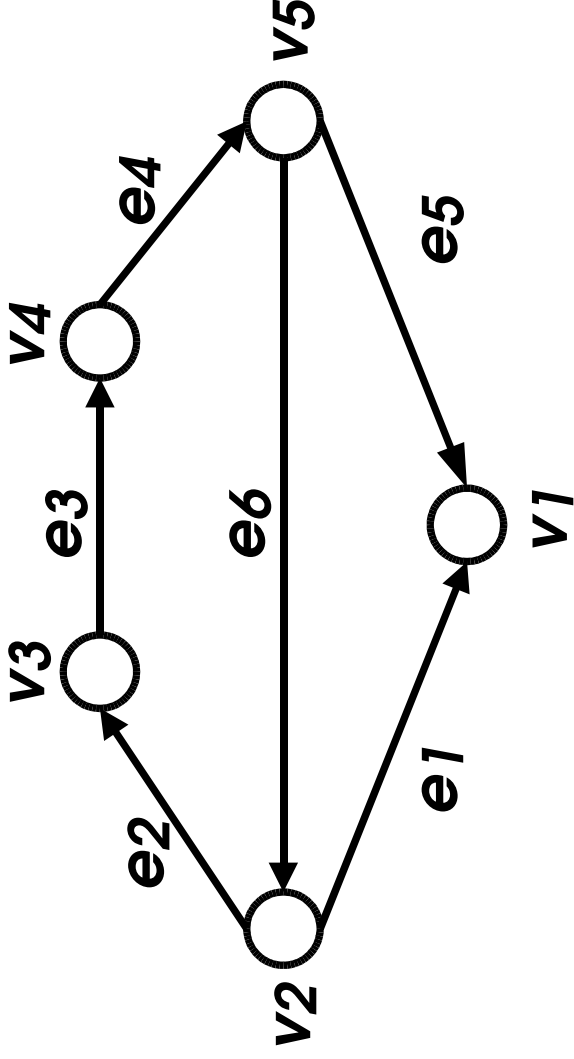
Operationen auf AG-Matrizen



0	0	0	0	0	0
1	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1
1	1	0	0	0	0

- Test, ob $(v_i, v_j) \in E$
- Nachsehen in A_{ij} : $O(1)$
- Welche v sind direkt mit u_i verbunden?
- Zeile i durchgehen: $O(n)$
- Ineffizient bei vielen Nullen

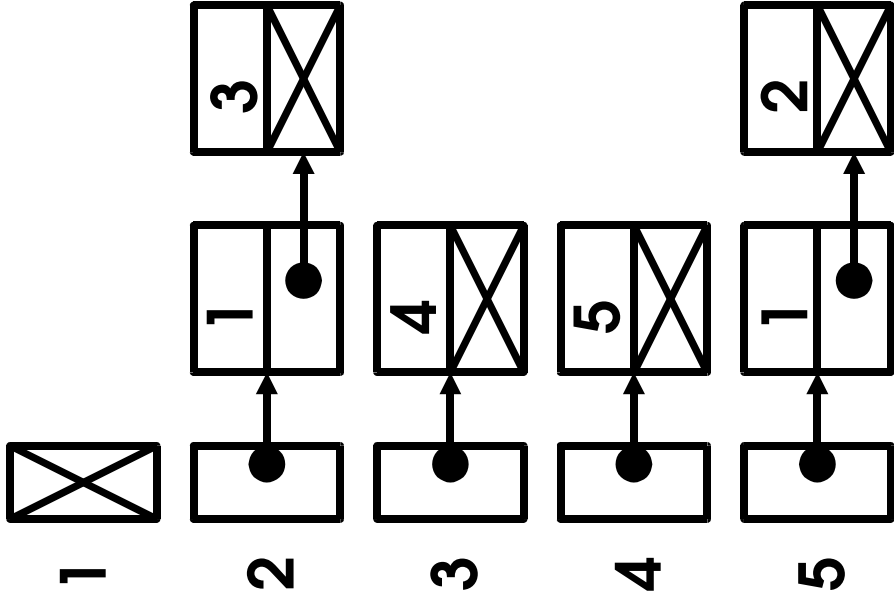
Adjanzlisten



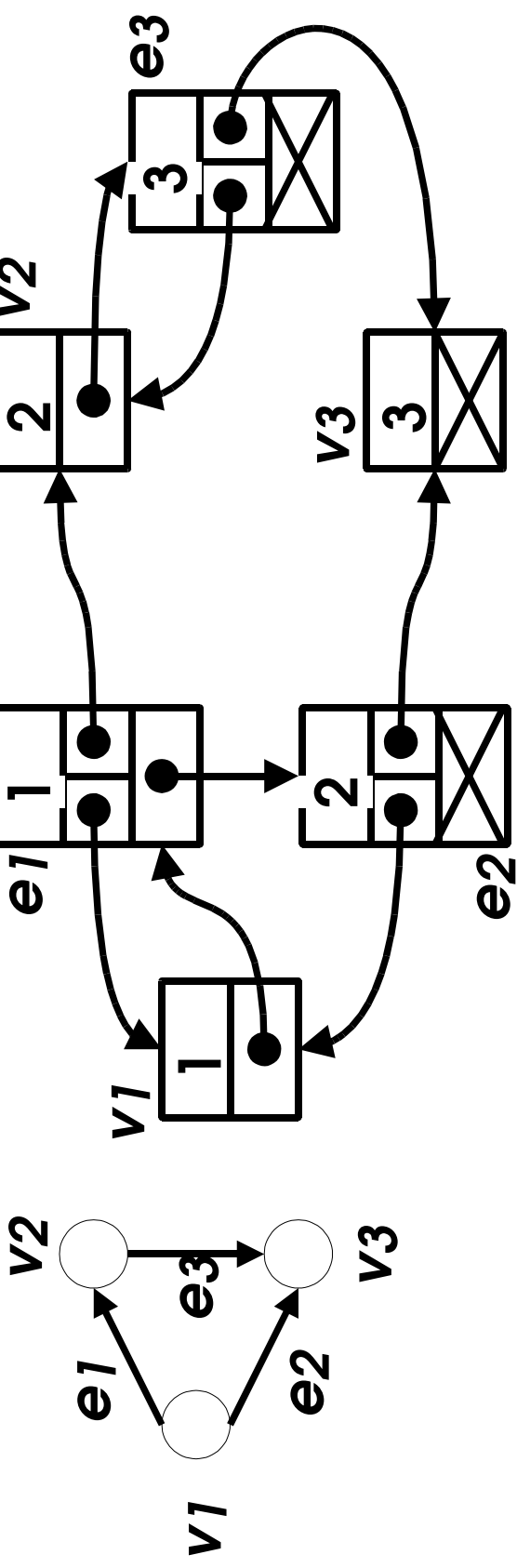
- Array aus Listen
- Knotennummer ist Index
- Listenelemente
- Index des Zielknotens
- Verkettung

Operationen auf Adjazenzlisten

- Test, ob $(v_i, v_j) \in E$
 - durchschnittlicher Außengrad: $k(G)$
 - $O(k)$
 - Unabhängig von n
- Welche v sind direkt mit u_j verbunden?
 - $O(k)$



Explizite Knoten und Kanten



■ Zugriff auf Knoten und Kanten

vertex_index	3
outgoing_edges	

■ Z.B. Gewichtung von

edge_index	3
from,	
to	
next	

- Knoten
- Kanten

Komplexitätstheorie

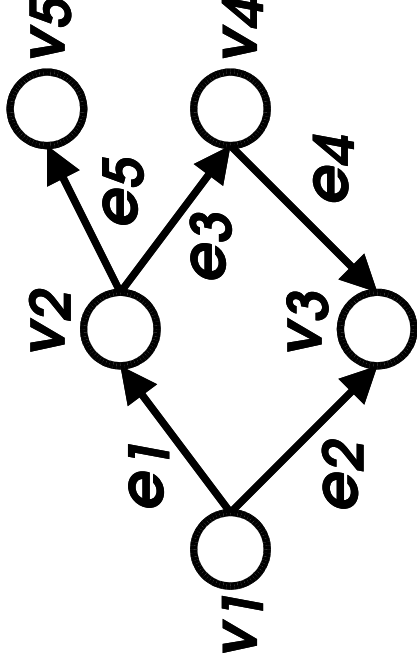
- O und Θ Notation
- Siehe Grundstudium!
- Wichtige Ordnungen
 - Exponentiell, z.B. 2^n .
 - Polynomial, z.B. n^3 .
 - Quadratisch, z.B. n^2 .
 - Logarithmisch, z.B. $n \log n$.
 - Linear, z.B. n .
 - Sublinear, z.B. 1.

Graphen durchlaufen

- **Aufgabe**
 - Besuche alle V und E von $G(V, E)$
 - Jedes Element genau einmal!
- **Unterschiedliche Reihenfolgen möglich**
- **Weit verbreitet**
 - **Tiefensuche**
 - ◆ Suche von Ursprungsknoten entfernen
 - **Breitensuche**
 - ◆ Erstmal angrenzende Knoten bearbeiten

Tiefensuche (DFS) - 1

```
dfs(vertex v) {  
  v.mark := 0;  
  v.process();  
  foreach (v,u) ∈ E {  
    (v,u).process();  
    if (u.mark) dfs(u);  
  }  
}  
main() {  
  foreach v ∈ V  
    v.mark := 1;  
  foreach v ∈ V  
    if (v.mark) dfs(v)  
}
```



```
dfs(v1)  
  (v1, v2)  
    dfs(v2)  
      (v2, v4)  
        dfs(v4)  
          (v4, v3)  
            dfs(v3)  
              (v2, v5)  
                dfs(v5)  
                  (v1, v3)
```

Tiefensuche (DFS) - 2

- **Komplexität für DFS auf $G(V,E)$**
 - Jeder Knoten einmal besucht
 - Jede Kante einmal besucht

→ $O(|V| + |E|)$
- **Anwendungsbeispiele**
 - Systematischer Graphdurchlauf
 - Finden der von einem Startknoten aus erreichbaren Knoten
 - ◆ Ersetze Schleife in main() durch einfachen Aufruf

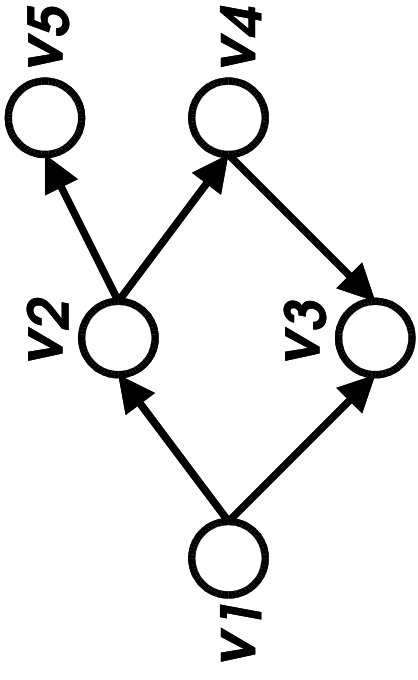
Breitensuche (BFS) - 1

```
bfs(vertex v) {  
  FIFO Q = ();  
  vertex u, w;  
  
  Q.shift_in(v);  
  do {  
    w := Q.shift_out();  
    w.process();  
    foreach (w,u) ∈ E do {  
      if (u.mark) {  
        u.mark := 0;  
        Q.shift_in(u);  
      }  
    }  
  } while (Q ≠ ())  
}
```

```
main() {  
  foreach v ∈ V do v.mark := 1;  
  foreach v ∈ V do  
    if (v.mark) {  
      v.mark := 0;  
      bfs(v);  
    }  
}
```

Breitensuche (BFS) - 2

```
dfs(vertex v) {  
  FIFO Q = ();  
  vertex u, w;  
  
  Q.shift_in(v);  
  do {  
    w := Q.shift_out();  
    w.process();  
    foreach (w,u) ∈ E do {  
      if (u.mark) {  
        u.mark := 0;  
        Q.shift_in(u);  
      }  
    }  
  } while (Q ≠ ())  
}
```



(v1)	v1	(v1,v2)
(v2)		(v1,v3)
(v2,v3)	v2	(v2,v4)
(v3,v4)		(v2,v5)
(v3,v4,v5)	v3	
(v4,v5)	v4	(v4,v3)
(v5)	v5	

Breitensuche (BFS) - 3

- **Komplexität für BFS auf $G(V,E)$**
 - Jeder Knoten einmal besucht
 - Jede Kante einmal besucht

→ $O(|V| + |E|)$
- **Anwendungsbeispiele**
 - Systematischer Graphdurchlauf
 - Finden der von einem Startknoten aus erreichbaren Knoten
 - Besuche Knoten in Reihenfolge der Entfernung (Pfadlänge) vom Startknoten

DFS und BFS

■ Weshalb die die äußeren Schleifen?

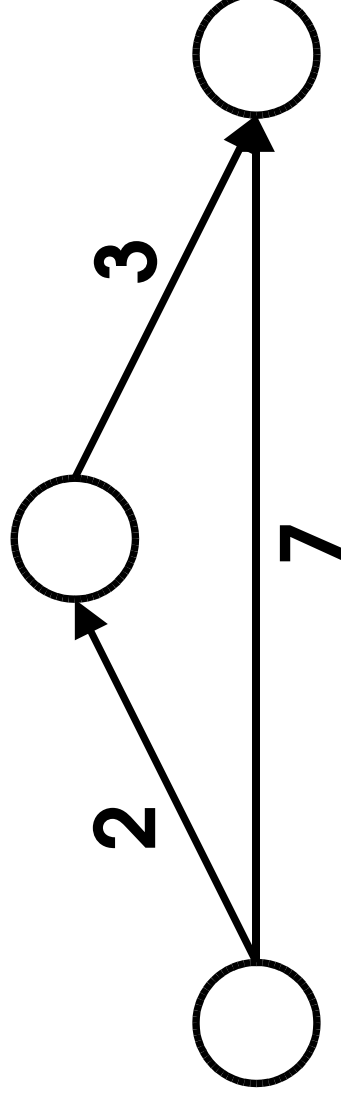
- Jeweils in `main()`
- Um `dfs(v)` bzw. `bfs(v)`

```
main() {  
  foreach v ∈ V  
    v.mark := 1;  
  foreach v ∈ V  
    if (v.mark) dfs(v)  
}
```

```
main() {  
  foreach v ∈ V do v.mark := 1;  
  foreach v ∈ V do  
    if (v.mark) {  
      v.mark := 0;  
      bfs(v);  
    }  
}
```

Kürzester Pfad

- Bestimme den kürzesten Pfad vom Startknoten zu den anderen Knoten
- Bei ungewichteten Graphen z.B. mit BFS
 - Erweitert um Verwaltung der Pfade
- ✘ Nicht bei gewichteten Graphen!
 - Niedrige Anzahl von Kanten nicht immer kürzester (leichtester) Weg



Kürzester Pfad nach Dijkstra - 1

dijkstra(**set**<vertex> V, vertex v_s , vertex v_t)

set<vertex> T; vertex u, v ;

$V := V \setminus \{v_s\}$; T := { v_s };

$v_s.dist := 0$;

foreach $u \in V$ **do**

if ($(v_s, u) \in E$)

then $u.dist := (v_s, u).weight$;

else $u.dist := +\infty$;

while ($v_t \notin T$) **do** {

$u := V.findmin(dist)$;

T := T \cup { u };

$V := V \setminus \{u\}$;

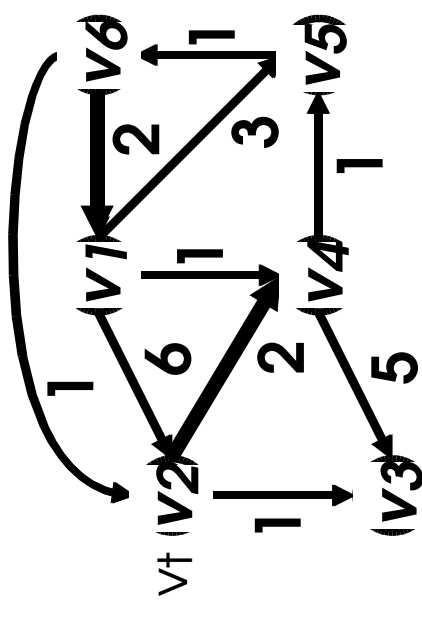
foreach ($u, v \in E$) **do**

if ($v.dist > u.dist + (u,v).weight$)

$v.dist := u.dist + (u,v).weight$;

}

}



T =

$v_i.dist$	$i=1$	2	3	4	5	6
{ v_1 }	0	6	∞	∞	∞	∞
{ v_1, v_4 }		6	6	6	2	∞
{ v_1, v_4, v_5 }		6	6	6	2	3
{ v_1, v_4, v_5, v_6 }		4	6	6	5	3
{ v_1, v_4, v_5, v_6, v_2 }						
{ $v_1, v_4, v_5, v_6, v_2, v_3$ }						

Kürzester Pfad nach Dijkstra -2

- Komplexität
 - while ($v \notin T$): $|V|$ -mal durchlaufen
 - ◆ $V.\text{findmin}(\text{dist})$: $O(|V|)$ je Suche
 - ↳ $O(|V|^2)$
 - foreach $(u,v) \in E$: $|E|$ -mal insgesamt
 - ◆ Einfacher Graph hat max. $|V|^2$ Kanten
 - ↳ $O(|V|^2)$
 - Gesamtaufwand $O(|V|^2 + |V|^2) = O(|V|^2)$

Nächste Veranstaltung

- **Vorlesung am Freitag**
- **Vorbereitungstipps**
 - Kapitel 6 und 7.1 lesen
 - Ggf. Kapitel 4 (Komplexität) wiederholen

Zusammenfassung

- **VLSI**
 - **Entwurfsbereiche**
 - **Tätigkeiten**
 - **Werkzeuge**
- **Hierarchie und Abstraktion**
- **Graphentheorie**
 - **Konzepte und Begriffe**
 - **Datenstrukturen**
 - **Algorithmen: DFS, BFS, SP**