Algorithmen für Chip-Entwurfswerkzeuge

Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesung
WS 2012/2013

Florian Stock, Andreas Koch

Eingebette Systeme und Anwendungen
Technische Universität Darmstadt

Embedded Systems & Applications

Material

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Vorlesung

TECHNISCHE UNIVERSITÄT DARMSTADT

- 3 CP
- Normale Prüfung zum Ende der Vorlesung
- Je nach Andrang mündlich oder schriftlich
  vermutlich mündlich, dann Länge ca. 30 Minuten
- Dringend empfohlen:
  Das begleitende Praktikum für 6 CP

Zeitplan und WWW

TECHNISCHE UNIVERSITÄT DARMSTADT

- Geplanter Zeitplan
  - Vorlesung:
    in KW 42-50,4,7 Dienstags und
    in den KW 43-45 zusätzlich Freitags
  - Praktikum:
    Kickoff (KW 42) und Vorträge
    jeweils Freitags (KW 46, 50, 4 und 7)
- Web-Seite
  - Fachgebiets-Webseite
  - Material und Ankündigungen

Fragen?

TECHNISCHE UNIVERSITÄT DARMSTADT

Noch Fragen zur Orga?

Überblick

TECHNISCHE UNIVERSITÄT DARMSTADT

- VLSI Entwurf
  - Probleme
  - Bereiche
  - Tätigkeiten
  - $\Rightarrow$ Werkzeuge
- Algorithmische Graphentheorie
  - Strukturen
  - Verfahren

VLSI Entwurfsproblem

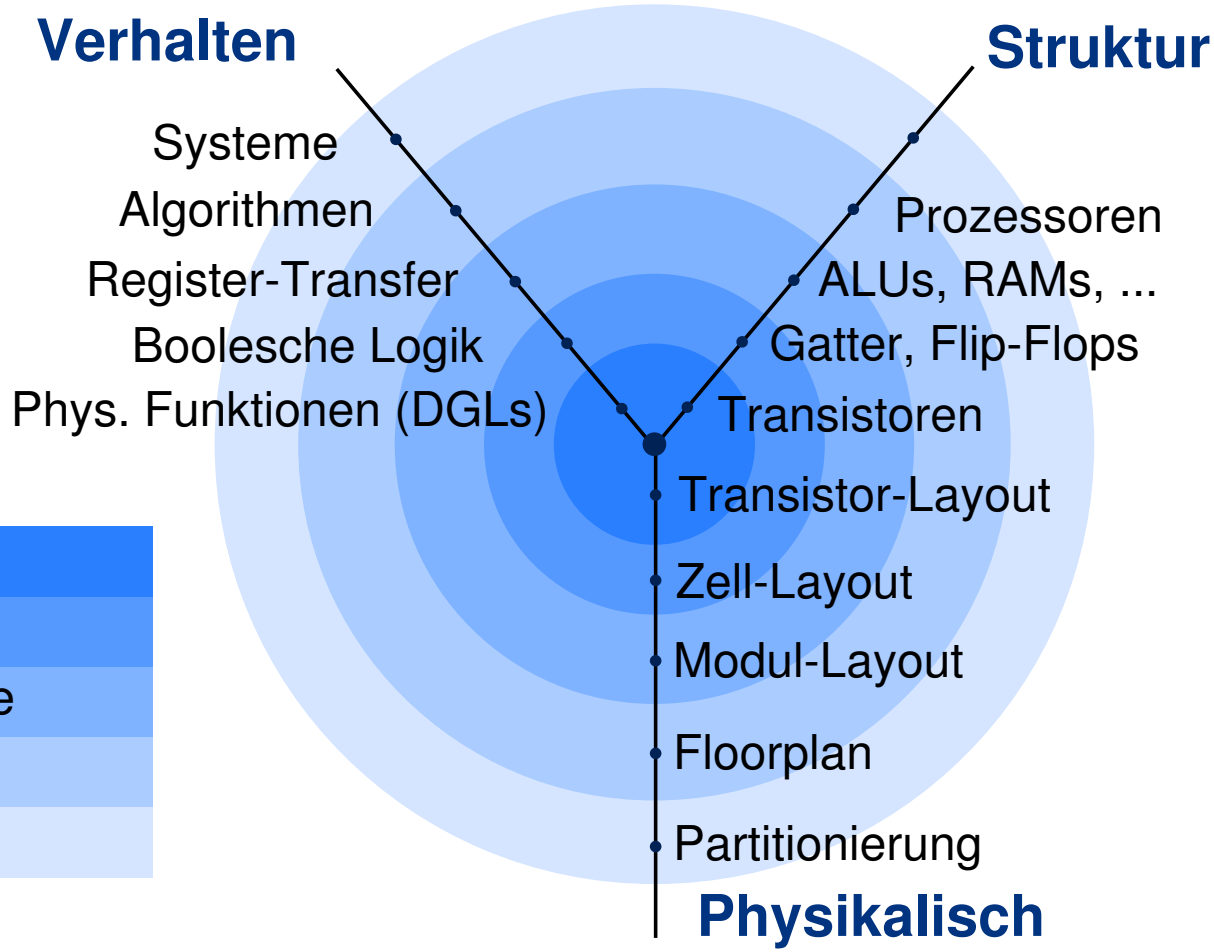
TECHNISCHE UNIVERSITÄT DARMSTADT

- “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

$\Rightarrow$ Aufteilen und vereinfachen
$\Rightarrow$ Qualitätseinbußen

Gajskis Y-Diagramm

TECHNISCHE UNIVERSITÄT DARMSTADT



Abstraktionsebenen

- Schaltkreisebene
- Logikebene
- Register-Transfer-Ebene
- Algorithmische Ebene
- Systemebene

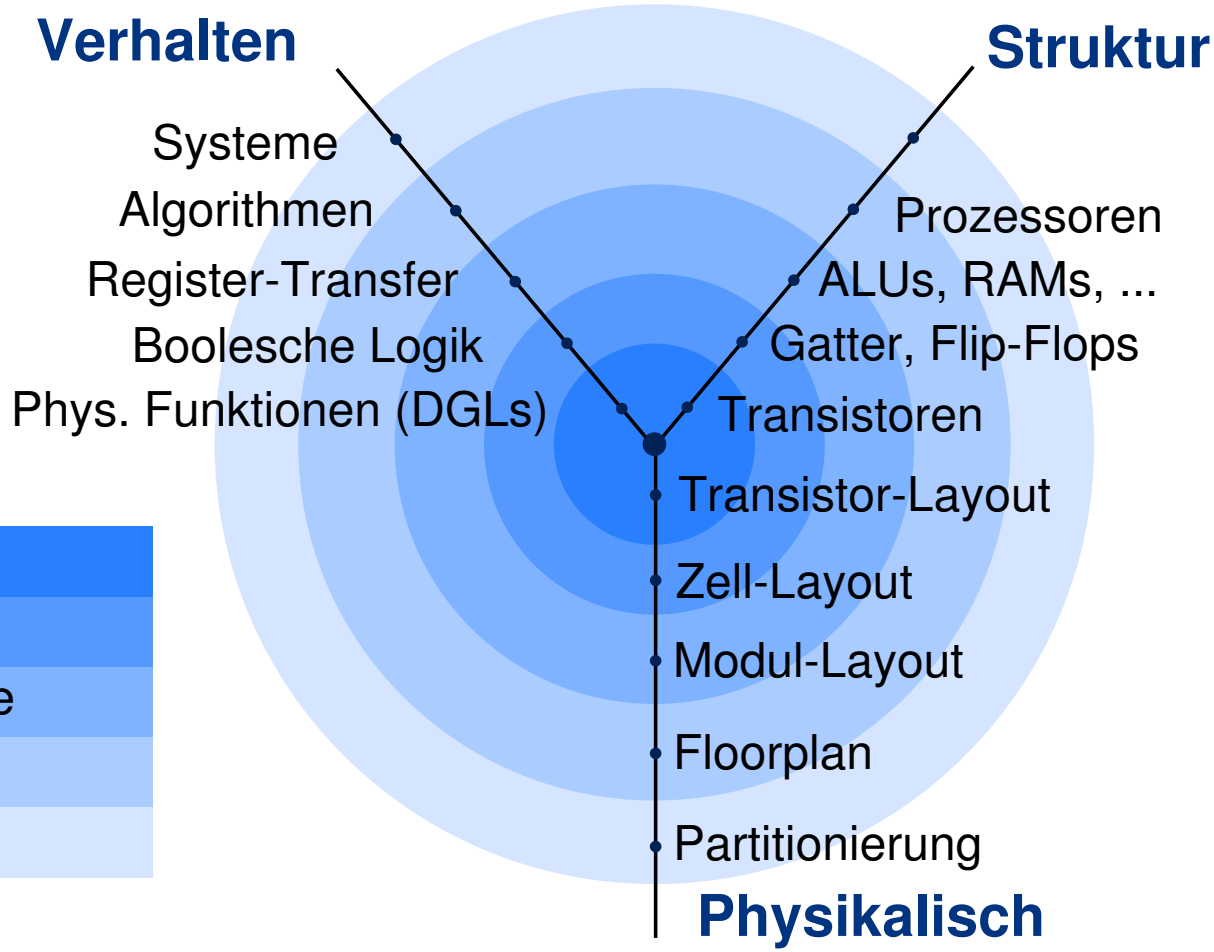
Tätigkeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Werkzeuge

TECHNISCHE UNIVERSITÄT DARMSTADT

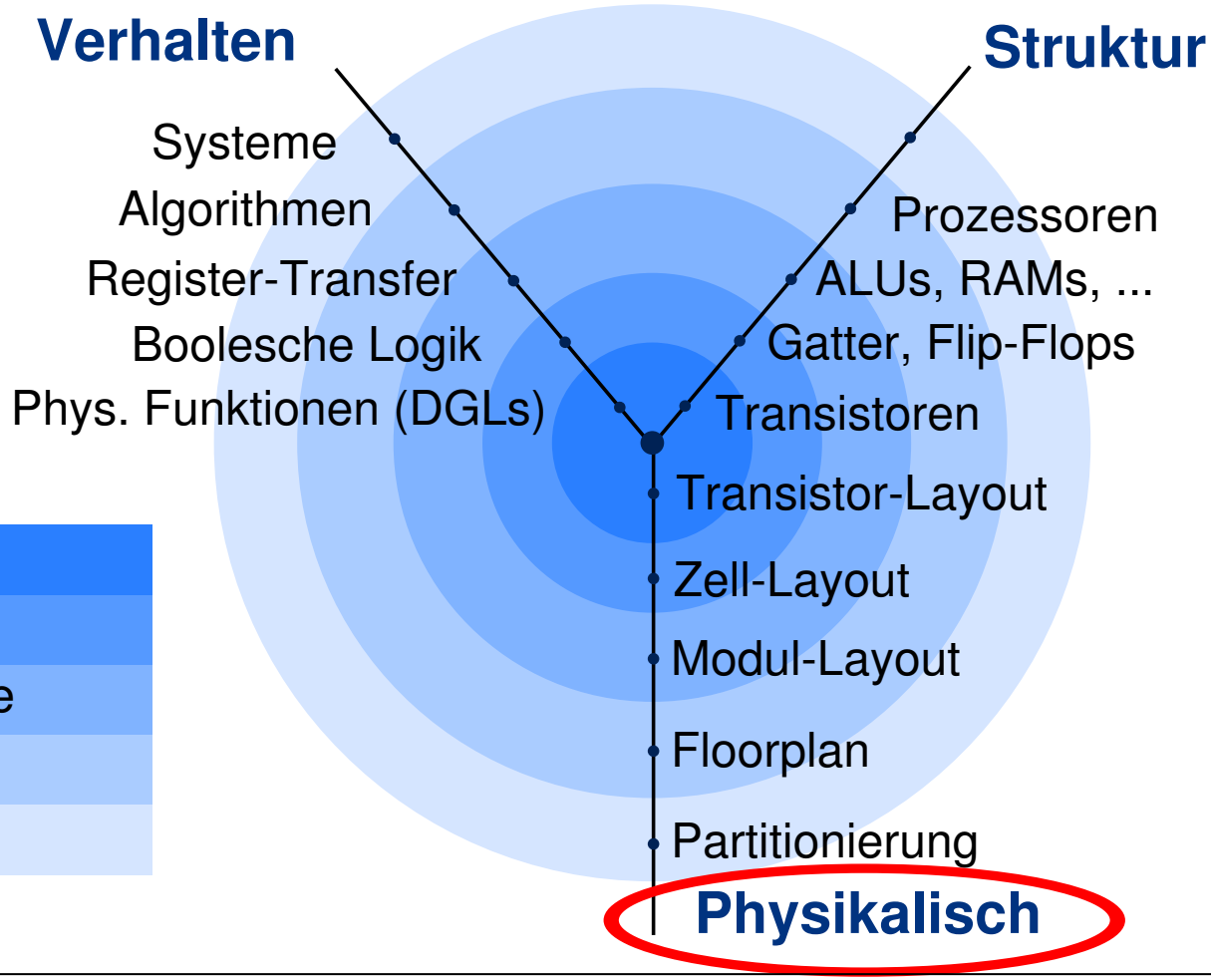


Abstraktionsebenen

- Schaltkreisebene
- Logikebene
- Register-Transfer-Ebene
- Algorithmische Ebene
- Systemebene

Werkzeuge

TECHNISCHE UNIVERSITÄT DARMSTADT



Abstraktionsebenen

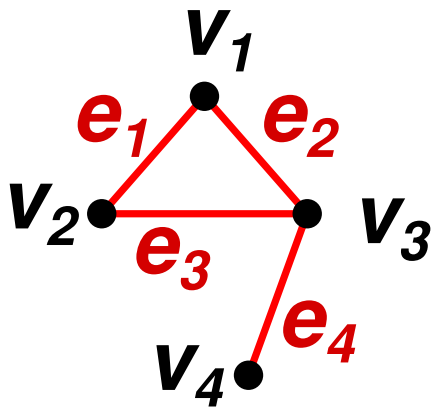
- Schaltkreisebene
- Logikebene
- Register-Transfer-Ebene
- Algorithmische Ebene
- Systemebene

(Ungerichteter) Graph

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph $G(V, E)$

- Eine Menge V von Knoten (vertex)
- Eine Menge E von Kanten (edge)
  - $e = \{v_1, v_2\}$
  - Kante e verbindet Knoten v_1 und v_2
  - Kante ist ein Menge mit 2 Elementen



Beispiel

$G = (V, E)$

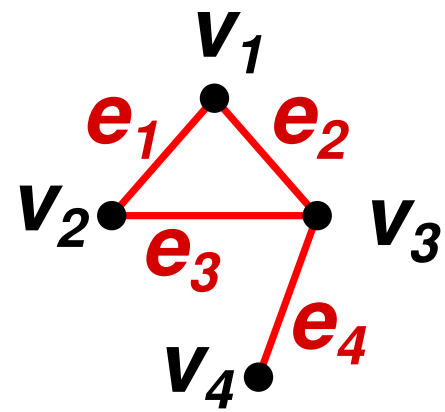
$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

$E = \{e_1, e_2, e_3, e_4, e_5\}$ mit $e_1 = \{v_1, v_2\}, e_2 = \{v_1, v_3\}, \ldots$

Knoteneigenschaften

TECHNISCHE UNIVERSITÄT DARMSTADT

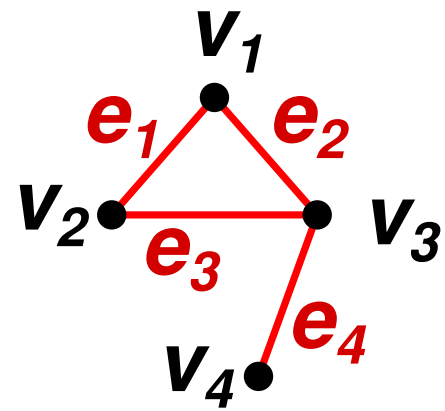
- $e = \{u, v\} \in E$
  - e ist inzident mit u (incident)
  - e ist inzident mit v (incident)
  - u ist adjazent mit v (adjacent)
- Grad $g(v) = |\{e \in E | v \in e\}|$ (degree)



Teilgraphen (Subgraphs)

TECHNISCHE UNIVERSITÄT DARMSTADT

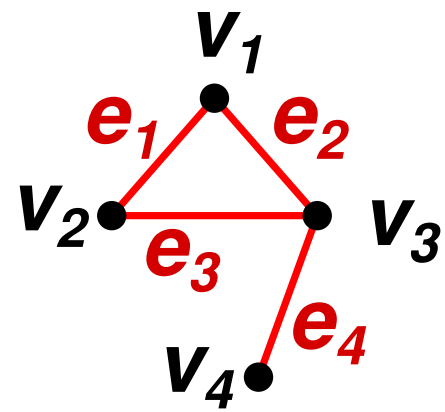
- $G(V, E)$ Graph
  - Graph $G'(V', E')$ heißt Teilgraph von G, falls
    - $V' \subseteq V$, und
    - $E' \subseteq E$, und
    - weiterhin gilt:
      $e = \{v_1, v_2\} \in E' \Rightarrow v_1, v_2 \in V'$
  - Anschaulich:
    - Entferne Knoten von G, und
    - Alle dazu inzidenten Kanten, und
    - Beliebige weitere Kanten
  - Werden nur die inzidenten Kanten entfernt:
    G' heisst dann (von Teilknotenmenge V')
    induzierter Teilgraph (induced subgraph)
- G heißt Supergraph zu G'



Vollständigkeit und Cliquen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Komplett untereinander verbundene Knoten bilden einen vollständigen Graph (complete graph) ($|E| = \frac{|V|(|V|-1)}{2}$)
- Induzierte Teilgraphen die vollständig sind heißen Cliquen.
- Cliquen die nicht Teilgraph von anderen Cliquen sind, heißen maximale Cliquen.



Erweiterungen und Spezialisierungen

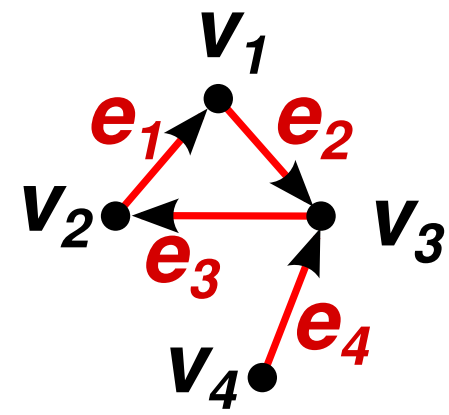
TECHNISCHE UNIVERSITÄT DARMSTADT

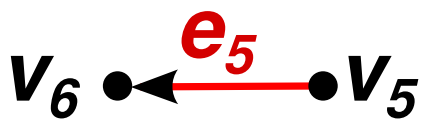
- In einfachen Graphen nicht zugelassen:
  - Schlingen (selfloop)
    Kante $\{u, v\}$ mit $u = v$
  - Parallele Kanten
    In Multigraphen erlaubt
  - Kanten e mit $|e| \neq 2$
    In Hypergraphen erlaubt
- Andere Erweiterungen:
  - Zusätzliche Gewichte an Knoten oder Kanten
    Gewichteter Graph (weighted graph)
  - Kanten, statt Menge, 2-Tupel: Gerichteter Graph (directed graph)

Gerichteter Graph
Definitionen

TECHNISCHE UNIVERSITÄT DARMSTADT

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

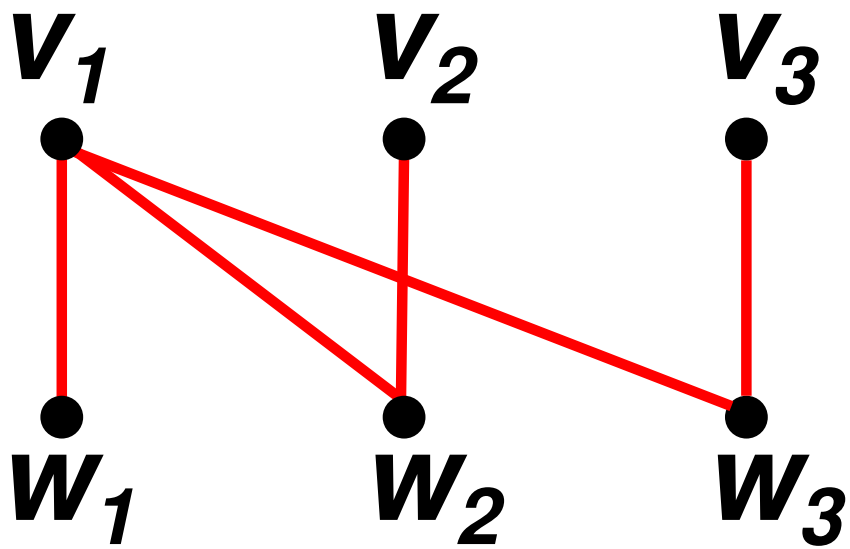




Spezielle Graphen
Bipartite Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kanten nur zwischen Knoten aus nichtüberlappenden Mengen
- $G = (V_1, V_2, E)$ ist bipartiter Graph
  - $V_1 \cap V_2 =$
  - $E = \{\{u, w\} | u \in V_1 \wedge w \in V_2\}$



Wege und Zyklen

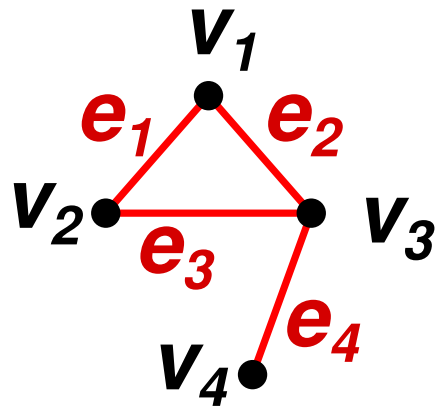
TECHNISCHE UNIVERSITÄT DARMSTADT

- Bei ungerichteten Graphen:

  Weg Geordnete Folge von Knoten und Kanten
  Beginnend und endend mit Knoten

  Länge Anzahl der Kanten

  Zyklus Anfang = Ende



- Bei gerichteten Graphen:

  Gerichteter Weg
  Gerichteter Zyklus

Wege und Zyklen

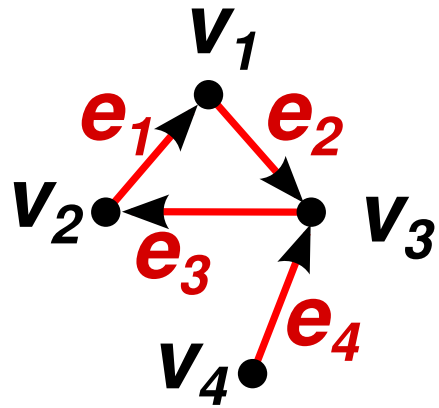
TECHNISCHE UNIVERSITÄT DARMSTADT

- Bei ungerichteten Graphen:

  Weg Geordnete Folge von Knoten und Kanten
  Beginnend und endend mit Knoten

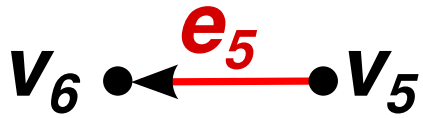
  Länge Anzahl der Kanten

  Zyklus Anfang = Ende



- Bei gerichteten Graphen:

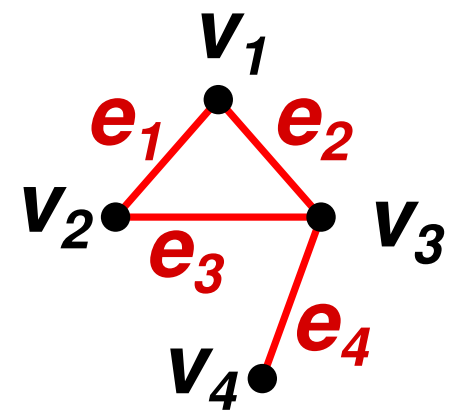
  Gerichteter Weg
  Gerichteter Zyklus



Zusammenhang
Ungerichteter Graph

TECHNISCHE UNIVERSITÄT DARMSTADT

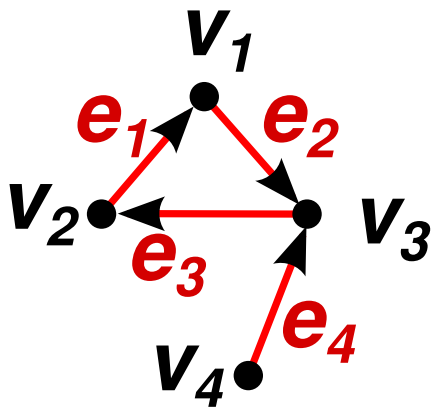
- *u* hängt mit *v* zusammen, :⇔
  Es gibt einen beide verbindenden Weg
- Zusammenhängender Graph:
  Alle Knoten hängen zusammen
- Zusammenhangskomponente
  Maximal zusammenhängende Teilgraphen

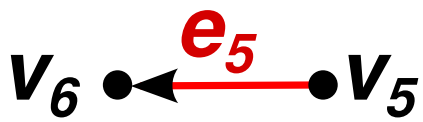


Zusammenhang
Gerichteter Graph

TECHNISCHE UNIVERSITÄT DARMSTADT

- Starker Zusammenhang von u und v zusammen, :⇔
  Es gibt gerichteten Weg von u nach v und von v nach u
- Stark zusammenhängende Komponenten:
  Alle enthaltenen Knoten hängen stark zusammen
- Schwacher Zusammenhang: Weg





Landau-Symbole

TECHNISCHE UNIVERSITÄT DARMSTADT

- $\mathcal{O}$ und Θ
  Siehe Grundstudium!
- $f \in \mathcal{O}(g)$
  f ist asymptotisch durch g beschränkt
- $f \in \Theta(g)$
  $f \in \mathcal{O}(g)$ und $g \in \mathcal{O}(f)$

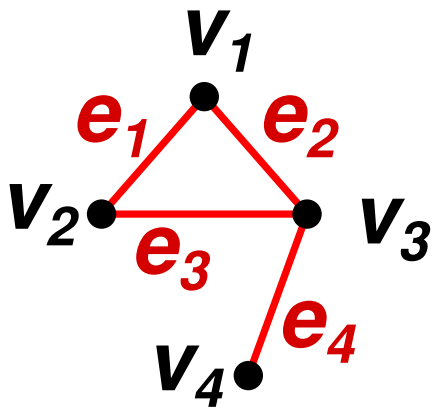
Wichtige Ordnungen

- Exponentiell, z.B. 2^n
- Polynomial, z.B. n^3
- Quadratisch, z.B. n^2
- Superlinear, z.B. $nlog(n)$
- Linear, z.B. n
- Sublinear, z.B. $log(n)$, $\sqrt{n}$
- Konstant, z.B. 1

Datenstrukturen
Adjazenzmatrix für Ungerichtete Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Adjazenzmatrix A_G 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
  - Statt 0 und 1 auch Gewichte möglich

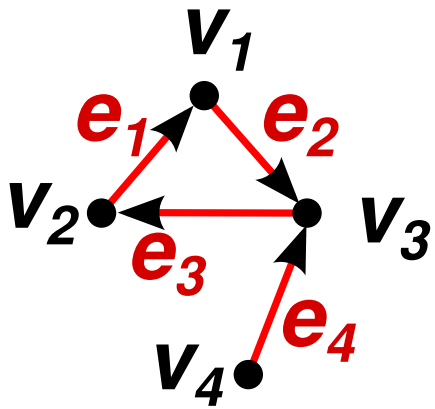


$$A_G = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

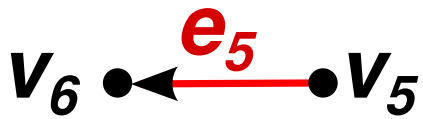
Datenstrukturen
Adjazenzmatrix für Gerichtete Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Adjazenzmatrix A_G von $G(V, E)$
  - $n \times n$ Matrix mit $n = |V|$
  - $A_{ij} = 1$ falls $(v_i, v_j) \in E$, sonst = 0
  - Matrix nicht mehr symmetrische
  - Statt 0 und 1 auch Gewichte möglich



$$A_G = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Datenstrukturen
Operationen auf Adjazenzmatrizen

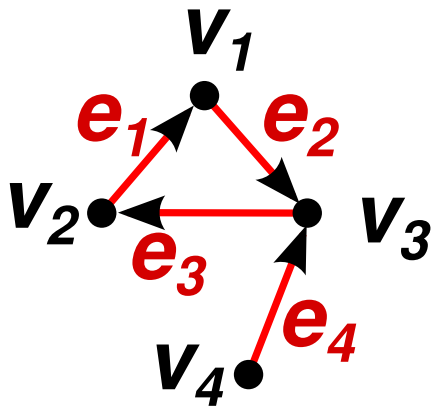
TECHNISCHE UNIVERSITÄT DARMSTADT

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

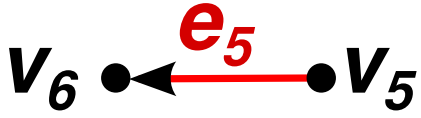
Datenstrukturen
(Knoten-Kanten-) Inzidenzmatrix

TECHNISCHE UNIVERSITÄT DARMSTADT

- Adjazenzmatrix A_G von $G(V, E)$
  - $m \times n$ Matrix mit $n = |V|, m = |E|$
  - Zeilen entsprechen Kanten, Spalten Knoten
  - Genau zwei nicht 0-Einträge pro Zeile
  - Bei gerichteten Graphen:
    $e_m = \{v_i, v_j\} \in E, A_{mj} = 1, A_{mi} = -1$
  - Bei ungerichteten Graphen:
    $e_m = \{v_i, v_j\} \in E, A_{mj} = 1, A_{mi} = 1$



$$A_G = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$



Datenstrukturen
Adjazenzliste

TECHNISCHE UNIVERSITÄT DARMSTADT

- Array aus Listen
  - Knotennummer ist Index
- Listenelemente
  - Index des Zielknotens
  - Verkettung
- Test, ob $(u, v) \in E$ unabhängig von n
  abhängig vom durchschnittlichen Außengrad k: $\mathcal{O}(k)$
- Kanten nur implizit gespeichert:
  Ggf. explizite Knoten- und Kantenmodellierung notwendig!

Graphen Travesierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

Graphen Travesierung
Tiefensuche (DFS) – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

```
dfs(vertex v)
begin
    v.mark := 0;
    v.process();
    foreach (v,u) ∈ E do
        (v,u).process();
        if (u.mark) then dfs(u);
    end
end
```

```
main()
begin
    foreach v ∈ V do
        v.mark := 1;
        foreach v ∈ V do
            if (v.mark) then dfs(v);
        end
    end
end
```

Graphen Travesierung
Tiefensuche (DFS) – Theoretisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

  $\Rightarrow \mathcal{O}(|V| + |E|)$
- Anwendungsbeispiele
  - Systematischer Graphdurchlauf
  - Finden der von einem Startknoten aus erreichbaren Knoten
    - Ersetze Schleife in `main()` durch einfachen Aufruf

Graphen Travesierung
Breitensuche (BFS) – Praktisch 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
bfs(vertex v)
begin
    FIFO Q := ();
    vertex u, w;
    Q.shift_in(v);
    repeat
        w := Q.shift_out();
        w.process();
        foreach (w,u) ∈ E do
            if (u.mark) then
                v.mark := 0;
                Q.shift_in(u);
            end
        end
    until Q == ();
end
```

```
main()
begin
    foreach v ∈ V do
        v.mark := 1;
        foreach v ∈ V do
            if (v.mark) then bfs(v);
        end
    end
end
```

Graphen Travesierung
Breitensuche (BFS) – Theoretisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

  $\Rightarrow \mathcal{O}(|V| + |E|)$
- Anwendungsbeispiele
  - Systematischer Graphdurchlauf
  - Finden der von einem Startknoten aus erreichbaren Knoten
  - Besuche Knoten in Reihenfolge der Entfernung vom Startknoten

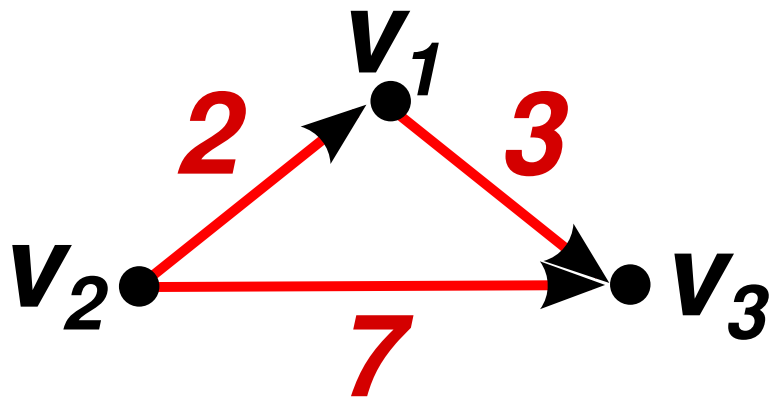
Graphen Travesierung
DFS und BFS

TECHNISCHE UNIVERSITÄT DARMSTADT

Kürzester Pfad

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bestimme den kürzesten Pfad vom Startknoten zu Zielknoten
  - Manchmal auch: zu allen 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



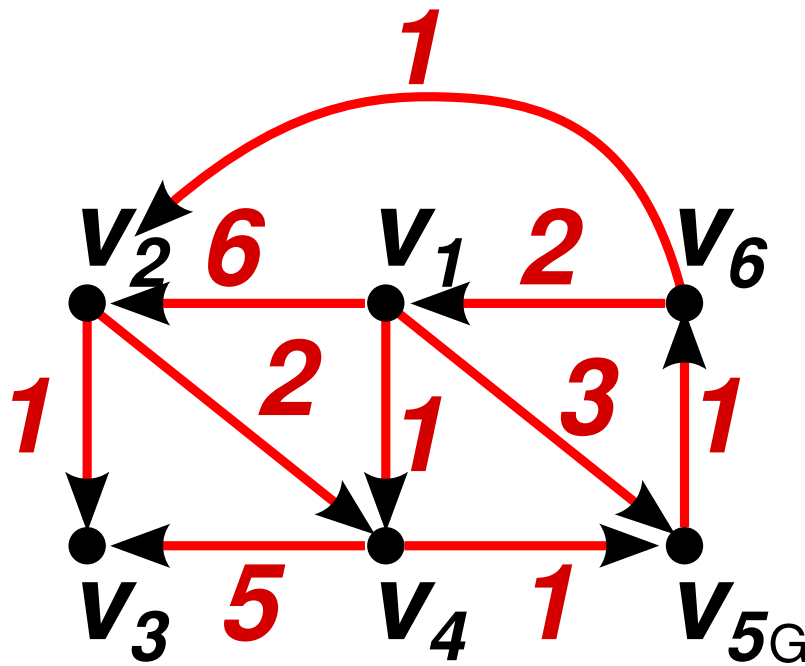
Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
set<vertex> T;
vertex u, v;
V := V - {vs };
T := { vs };
vs.dist := 0;
```
foreach $u \in V$ **do**
  if $((v_s, u) \in E)$ **then**
    `u.dist := (vs, u).weight;`
  else `u.dist := +∞;`
end

⋮



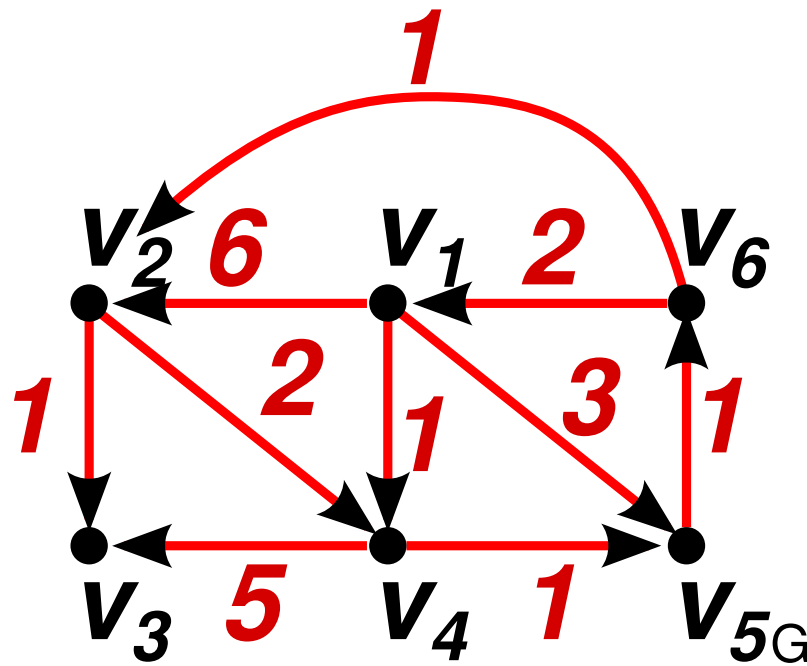
Kürzester Pfad von v_1 nach v_3

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

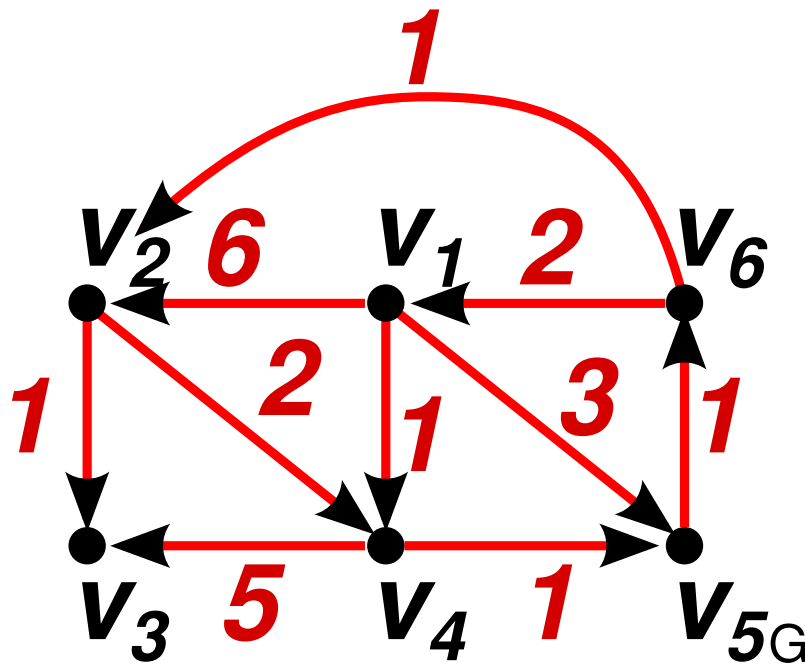
Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
set<vertex> T;
vertex u, v;
V := V - {v_s};
T := { v_s };
v_s.dist := 0;
foreach u ∈ V do
    if ((v_s, u) ∈ E) then
        u.dist := (v_s, u).weight;
    else u.dist := +∞;
end
```

⋮



Kürzester Pfad von v_1 nach v_3

T= $\{v_1\}$

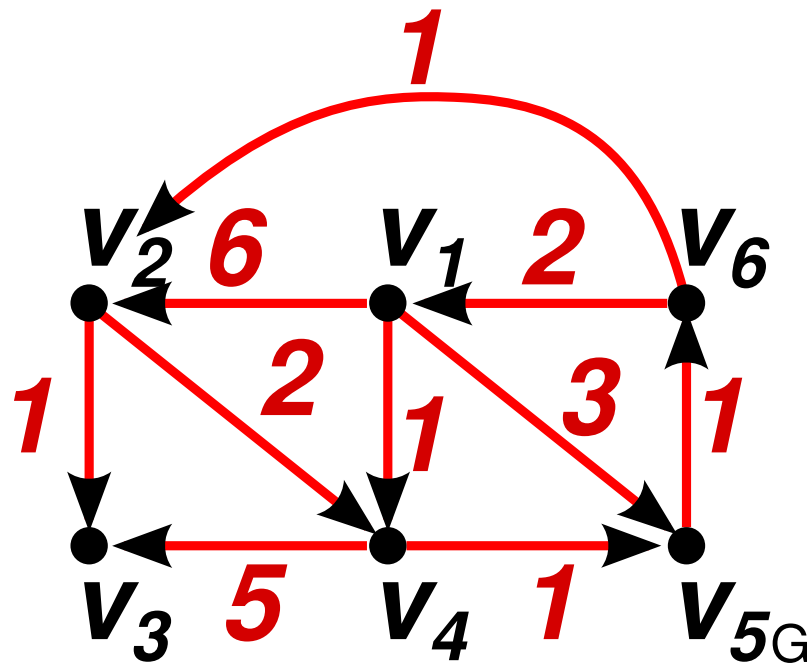
$v_i.dist$ = 0 6 ∞ 1 3 ∞

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1\}$

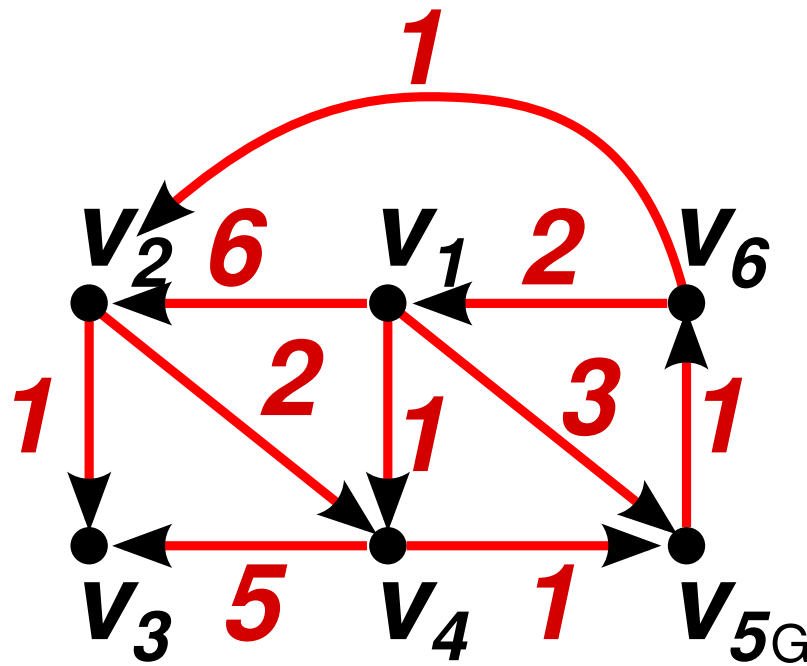
$v_i.dist$ = 0 6 ∞ 1 3 ∞

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

$T = \{v_1, v_4\}$

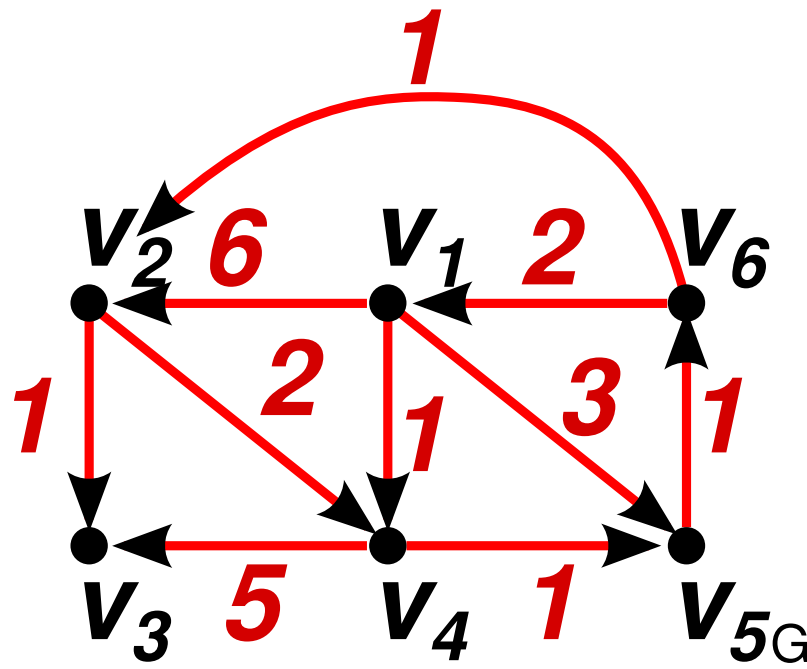
$v_i.dist = \quad 0 \quad 6 \quad \infty \quad 1 \quad 3 \quad \infty$

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4\}$

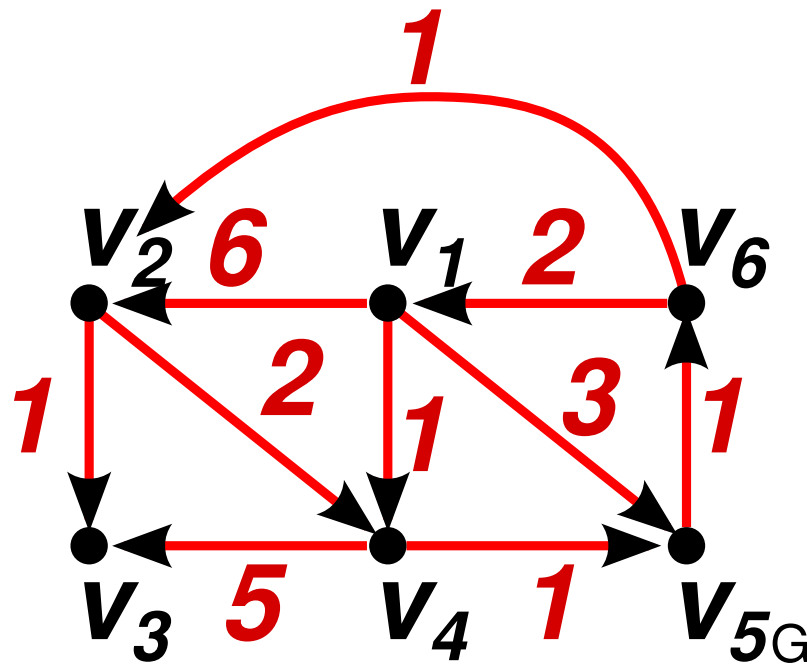
v_i.dist =	0	6	6	1	2	∞

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4, v_5\}$

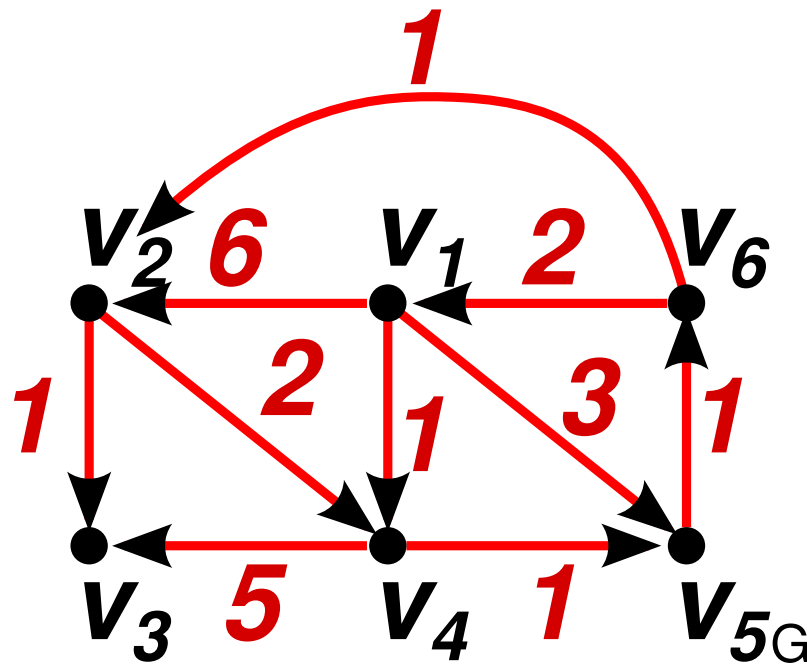
v_i.dist =	0	6	6	1	2	∞

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4, v_5\}$

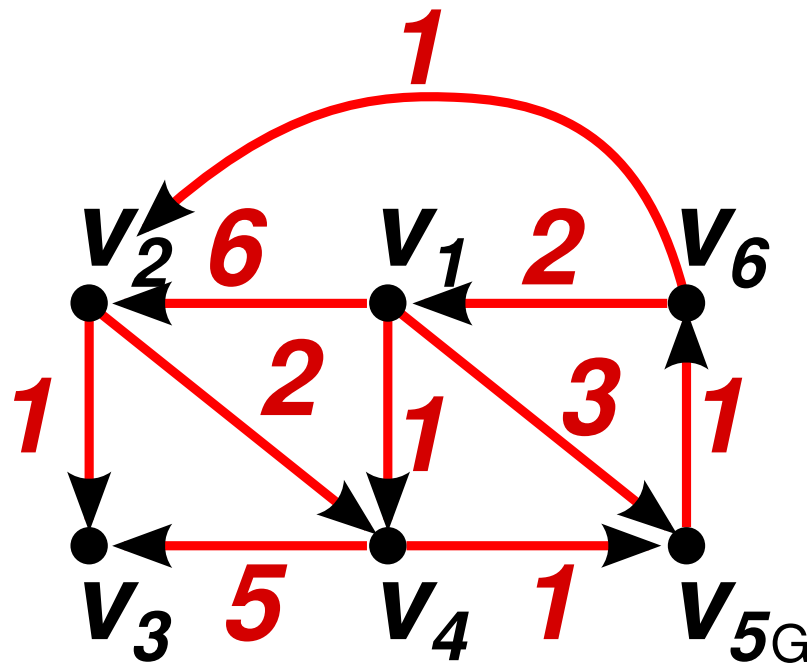
$v_i.dist$ = 0 6 6 1 2 3

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4, v_5, v_6\}$

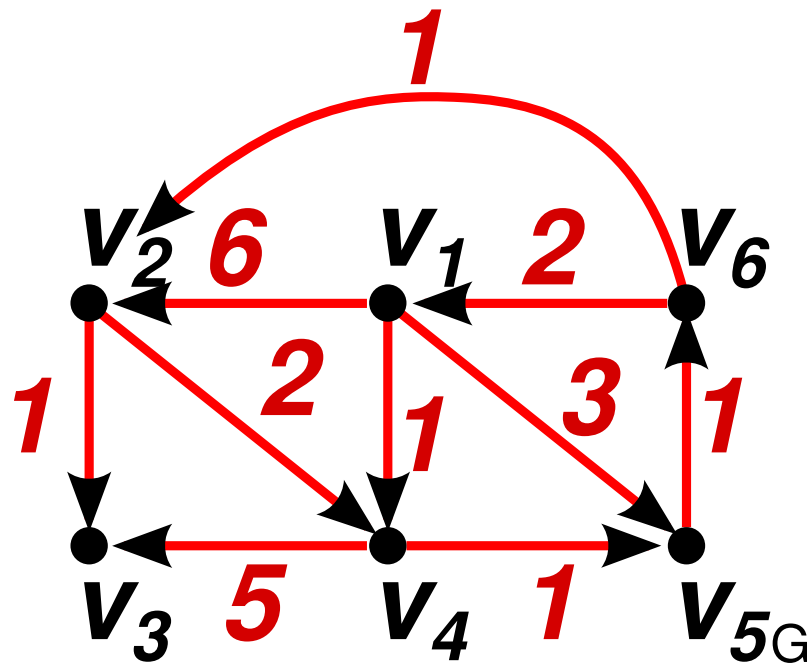
v_i.dist =	0	6	6	1	2	3

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4, v_5, v_6\}$

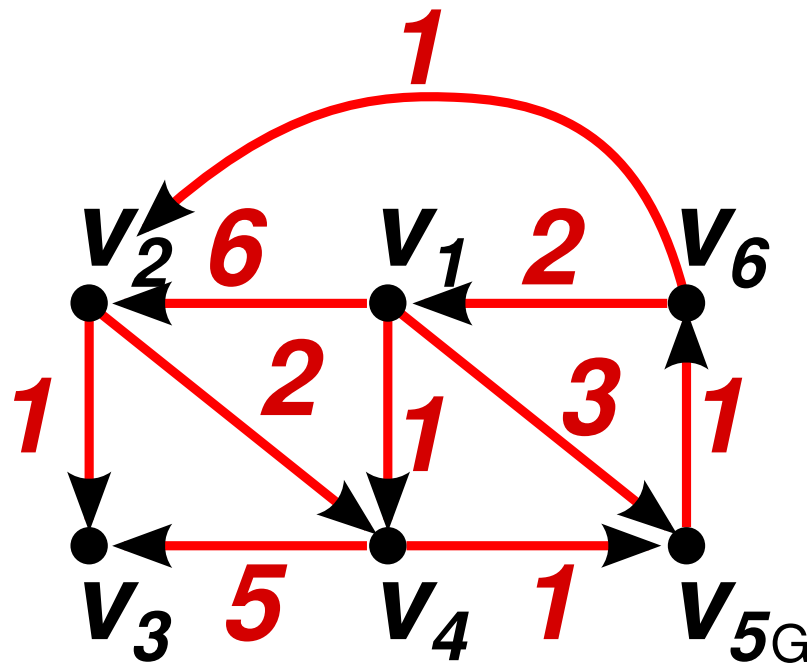
$v_i.dist$ = 0 4 6 1 2 3

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4, v_5, v_6, v_2\}$

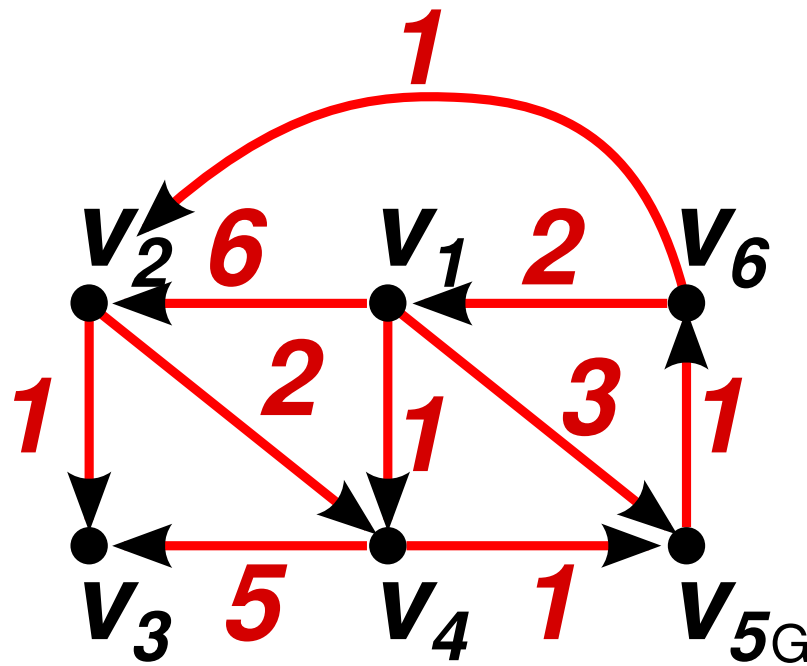
$v_i.dist$ = 0 4 6 1 2 3

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4, v_5, v_6, v_2\}$

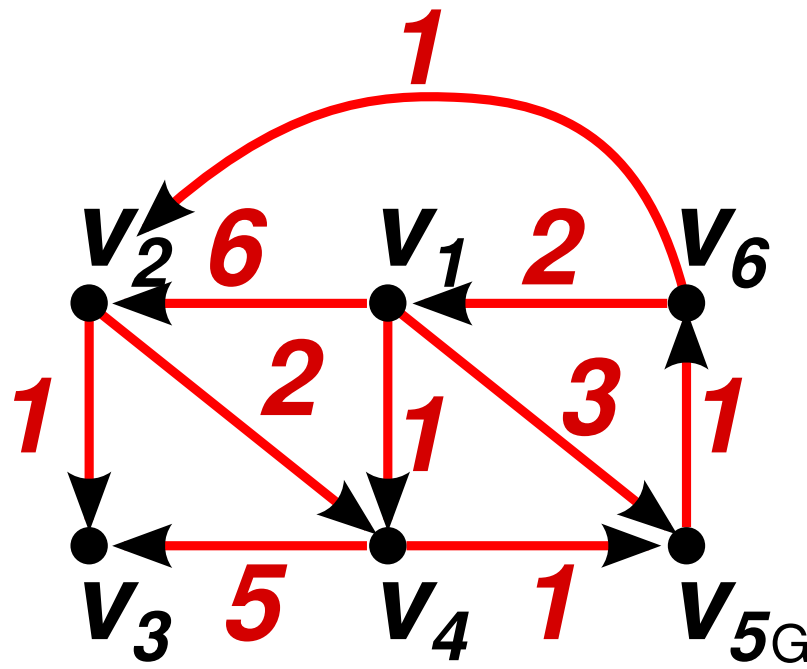
$v_i.dist$ = 0 6 5 1 2 3

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
⋮
while (v_t ∉ T) do
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
    foreach (u,v) ∈ E do
        if (v.dist > u.dist + (u, v).weight) then
            v.dist := u.dist + (u,v).weight;
    end
end
```



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4, v_5, v_6, v_2, v_3\}$

$v_i.dist$ = 0 6 5 1 2 3

Kürzester Pfad
Dijkstra – Theoretisch

TECHNISCHE UNIVERSITÄT DARMSTADT

- Komplexität
  - `while (v` $\notin$ `T)`: $|V|$-mal durchlaufen
    - `V.findmin(dist)`: $\mathcal{O}(|V|)$ je Suche

    $\Rightarrow$ $\mathcal{O}(|V|^2)$
  - `foreach (u,v)` $\in$ `E`: $|E|$-mal insgesamt
    - Einfacher Graph hat max. $|V|^2$ Kanten

    $\Rightarrow$ $\mathcal{O}(|V|^2)$
  - Gesamtaufwand $\mathcal{O}(|V|^2 + |V|^2) = \mathcal{O}(|V|^2)$

Nächste Veranstaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kickoff für das Praktikum am Freitag
- Nächste Vorlesung nächsten Dienstag
- Vorbereitungstipps
  - Kapitel 6 und 7.1 lesen
  - Ggf. Kapitel 4 (Komplexität) wiederholen

Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

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