Algorithmen für Chip-Entwurfswerkzeuge

Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesung
WS 2014/2015

Andreas Koch

Eingebettete Systeme und Anwendungen
Technische Universität Darmstadt

Embedded Systems & Applications

Material

- Grundlage der Vorlesung
  - *Algorithms for VLSI Design Automation*
    Sabih H. Gerez, Wiley & Sons, 1998
  - *Electronic Design Automation*
    L.-T. Wang, Y.-W. Chang & K-T. Cheng, Morgan-Kaufmann, 2009
- 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
  falls mündlich, Länge ca. 30 Minuten
- Dringend empfohlen:
  Das begleitende Praktikum für 6 CP

Zeitplan und WWW

- Geplanter Zeitplan
  - Vorlesung:
    Dienstags und
    in den ersten Wochen zusätzlich Freitags
  - Praktikum:
    Blockpraktikum, beginnt je nach Stofffortschritt
    Erwartet: Anfang in zweiter Semesterhälfte, Ende in vorlesungsfreier Zeit
- Web-Seite
  - Fachgebiets-Webseite
  - Material und Ankündigungen

Fragen?

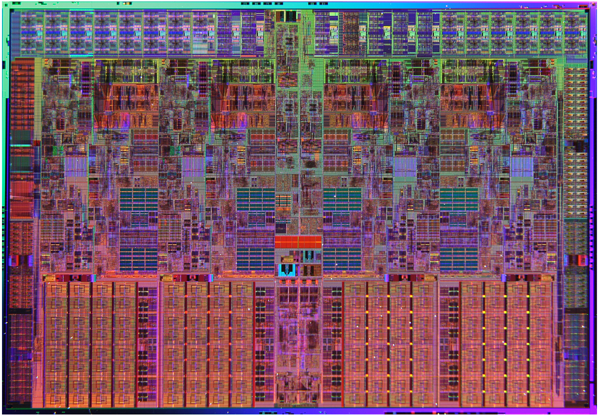
TECHNISCHE UNIVERSITÄT DARMSTADT

Noch Fragen zur Orga?

Überblick
Vorlesungsinhalte

- VLSI Entwurf
  - Probleme
  - Bereiche
  - Tätigkeiten
  - ⇒ Werkzeuge
- Algorithmische Graphentheorie
  - Strukturen
  - Verfahren
- Mathematische Optimierungsverfahren
  - Heuristische Algorithmen
  - Exakte Algorithmen

Ziel
Physikalischer Schaltkreis



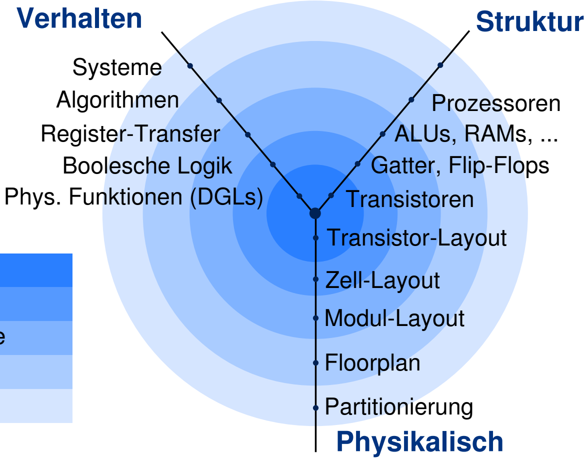
Quelle: Intel (Nehalem)

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
  manche Ziele auch diametral

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

Gajskis Y-Diagramm



Abstraktionsebenen

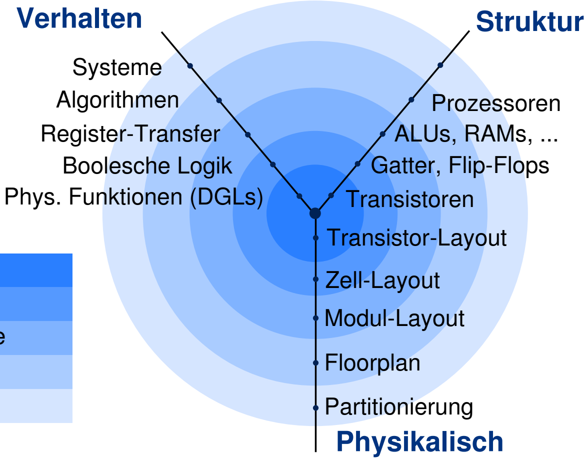
- Schaltkreisebene
- Logikebene
- Register-Transfer-Ebene
- Algorithmische Ebene
- Systemebene

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

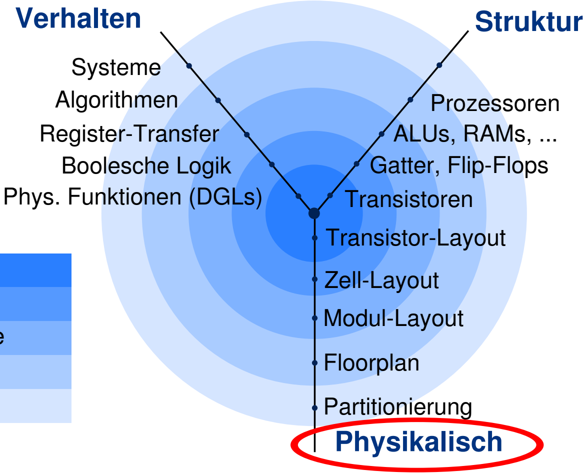
Werkzeuge



Abstraktionsebenen

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

Werkzeuge



Abstraktionsebenen

Schaltkreisebene
Logikebene
Register-Transfer-Ebene
Algorithmische Ebene
Systemebene

Ablauf-Vorlesung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Basis: Algorithmische Grundlagen
- Layout-Synthese-Algorithmen entsprechend dem Hardware-Entwicklungsfluß
  $\cdots \rightarrow$ Entwurf $\rightarrow$ Layout-Synthese $\rightarrow$ Layout-Verifikation $\rightarrow$ Fertigung $\rightarrow \cdots$
- Layoutsynthese:
  1. Partitionierung
  2. Floorplanning
  3. Platzierung
  4. Verdrahtung
  5. Kompaktierung

Grundlagen

- *Komplexitätstheorie*
- Graphen
  - Standardgraphen und Varianten
  - Datenstrukturen
  - Algorithmen
- Darstellungen von Schaltungen
- Optimierungsverfahren

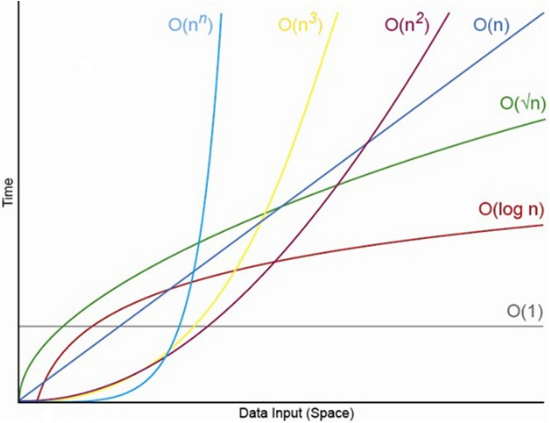
Komplexitätsklassen
Landau-Symbole

- $\mathcal{O}$ 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 $g \in \mathcal{O}(f)$
- Üblicherweise für Laufzeit benutzt kann aber auch für Speicher benutzt werden
  ($TIME \subseteq SPACE$)

Wichtige Ordnungen

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

Komplexitätsklassen
Vergleich



(Ungerichteter) Graph

Graph $G(V, E)$

- Eine Menge V von Knoten (vertex)

(Ungerichteter) Graph

Graph $G(V, E)$

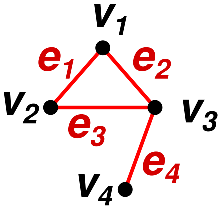
- Eine Menge V von Knoten (vertex)
- Eine Menge E von Kanten (edge)
  - $e = \{v_1, v_2\}$
  - Kante e verbindet Knoten v_1 und v_2
  - Kante ist ein Menge mit 2 Elementen

(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 $e_1 = \{v_1, v_2\}, e_2 = \{v_1, v_3\}, \ldots$

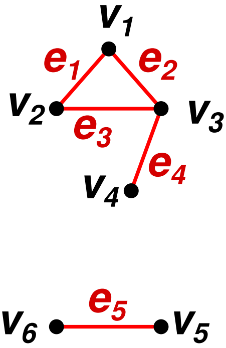
Knoteneigenschaften

TECHNISCHE UNIVERSITÄT DARMSTADT

- $e = \{u, v\} \in E$

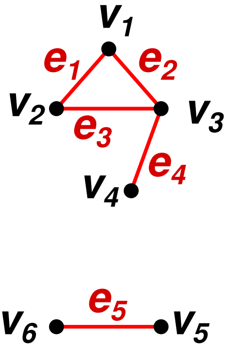
Knoteneigenschaften

- $e = \{u, v\} \in E$
  - e ist inzident mit u (incident)



Knoteneigenschaften

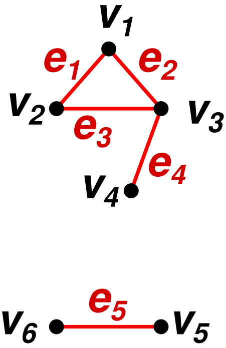
- $e = \{u, v\} \in E$
  - e ist inzident mit u (incident)
  - e ist inzident mit v (incident)



Knoteneigenschaften

TECHNISCHE UNIVERSITÄT DARMSTADT

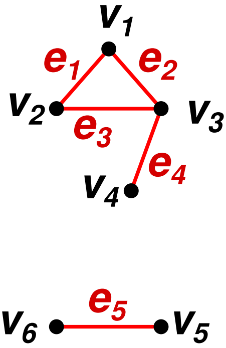
- $e = \{u, v\} \in E$
  - e ist inzident mit u (incident)
  - e ist inzident mit v (incident)
  - u ist adjazent mit v (adjacent)



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)

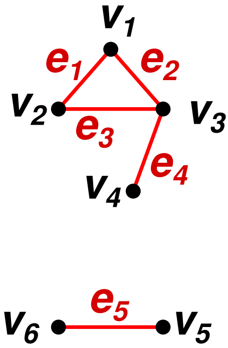
TECHNISCHE UNIVERSITÄT DARMSTADT

- $G(V, E)$ Graph

Teilgraphen (Subgraphs)

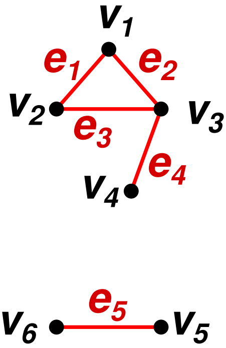
TECHNISCHE UNIVERSITÄT DARMSTADT

- $G(V, E)$ Graph
  - Graph $G'(V', E')$ heißt Teilgraph von G, falls
    - $V' \subseteq V$, und
    - $E' \subseteq E$, und
    - weiterhin gilt: $e = \{v_1, v_2\} \in E' \Rightarrow v_1, v_2 \in V'$



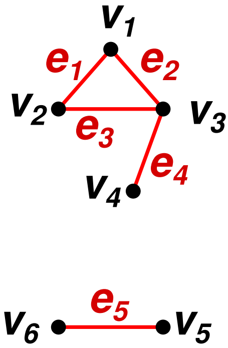
Teilgraphen (Subgraphs)

- $G(V, E)$ 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



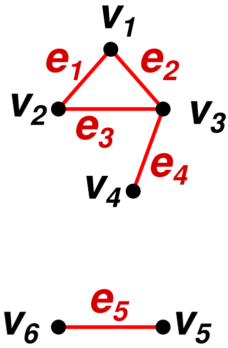
Teilgraphen (Subgraphs)

- $G(V, E)$ 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 oder Untergraph
    (induced subgraph)



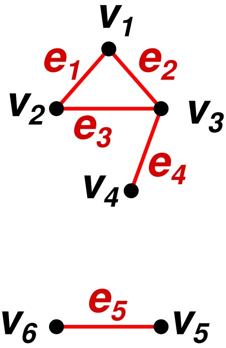
Teilgraphen (Subgraphs)

- $G(V, E)$ 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 oder Untergraph (induced subgraph)
- G heißt Supergraph zu G'



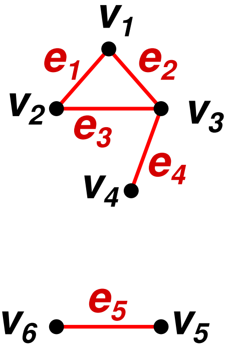
Vollständigkeit und Cliquen

- Komplett untereinander verbundene Knoten bilden einen vollständigen Graph (complete graph) ($|E| = \frac{|V|(|V|-1)}{2}$)



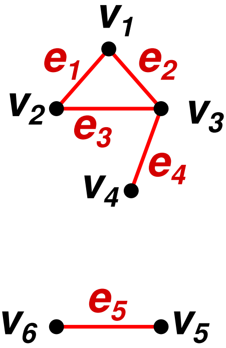
Vollständigkeit und Cliquen

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



Vollständigkeit und Cliquen

- 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

- In einfachen Graphen nicht zugelassen:

Erweiterungen und Spezialisierungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- In einfachen Graphen nicht zugelassen:
  - Schlingen (selfloop)
    Kante $\{u, v\}$ mit $u = v$

Erweiterungen und Spezialisierungen

- In einfachen Graphen nicht zugelassen:
  - Schlingen (selfloop)
    Kante $\{u, v\}$ mit $u = v$
  - Parallele Kanten
    In Multigraphen erlaubt

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

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:

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)

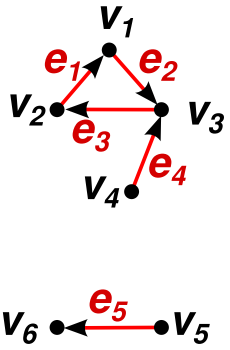
Erweiterungen und Spezialisierungen

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 sind 2-Tupel (Paare) statt Menge: Gerichteter Graph (directed graph)

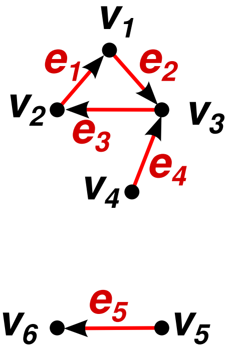
Gerichteter Graph
Definitionen

- $G(V, E)$ mit $e = (u, v)\ u, v \in E$
  - e inzident von u (ausgehend)



Gerichteter Graph
Definitionen

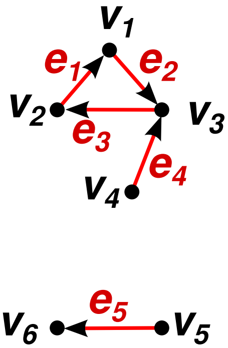
- $G(V, E)$ mit $e = (u, v)$ $u, v \in E$
  - e inzident von u (ausgehend)
  - e inzident nach v (eingehend)



Gerichteter Graph
Definitionen

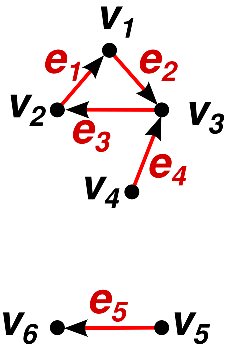
TECHNISCHE UNIVERSITÄT DARMSTADT

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



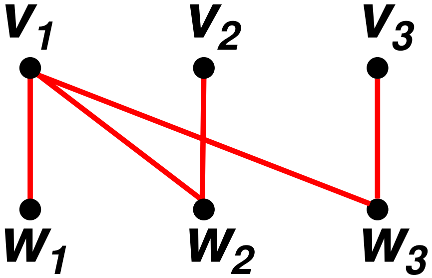
TECHNISCHE UNIVERSITÄT DARMSTADT

- $G(V, E)$ 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

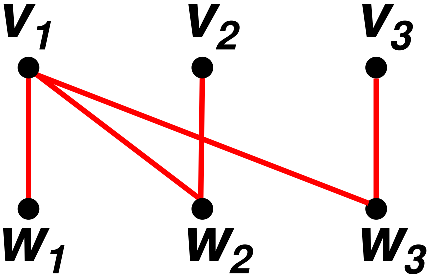
- Kanten nur zwischen Knoten aus nicht-überlappenden Mengen



Spezielle Graphen
Bipartite Graphen

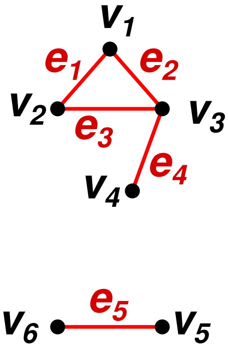
TECHNISCHE UNIVERSITÄT DARMSTADT

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



Wege und Zyklen

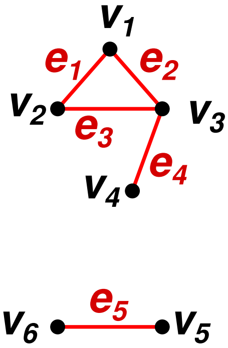
- Bei ungerichteten Graphen:



Wege und Zyklen

- Bei ungerichteten Graphen:

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

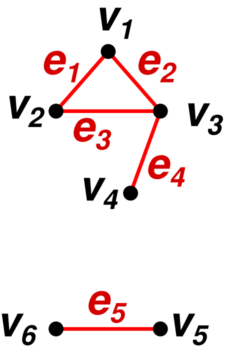


Wege und Zyklen

- Bei ungerichteten Graphen:

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

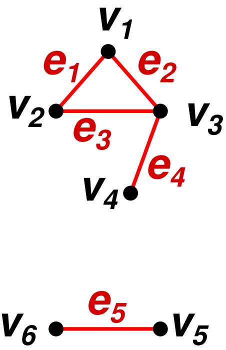
  Länge Anzahl der Kanten



Wege und Zyklen

- Bei ungerichteten Graphen:

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



Wege und Zyklen

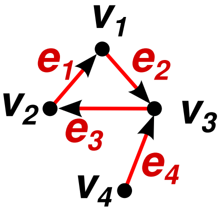
- Bei ungerichteten Graphen:

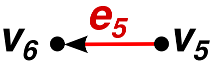
  Weg Geordnete Folge von Knoten und Kanten
  Beginnend und endend mit Knoten

  Länge Anzahl der Kanten

  Zyklus Anfang = Ende

- Bei gerichteten Graphen:

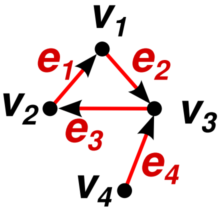




Wege und Zyklen

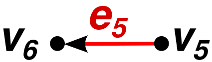
- Bei ungerichteten Graphen:

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



- Bei gerichteten Graphen:

  Gerichteter Weg



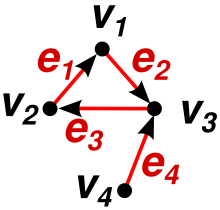
Wege und Zyklen

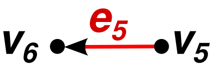
- Bei ungerichteten Graphen:

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

- Bei gerichteten Graphen:

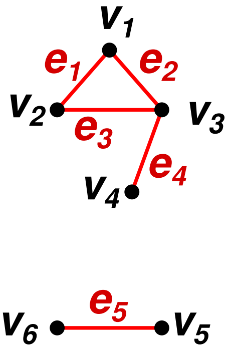
  Gerichteter Weg
  Gerichteter Zyklus





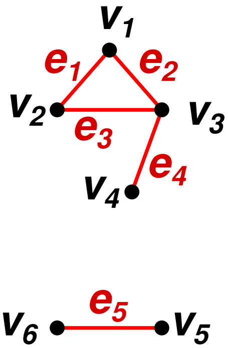
Zusammenhang
Ungerichteter Graph

- *u* hängt mit *v* zusammen, :⇔ Es gibt einen beide verbindenden Weg



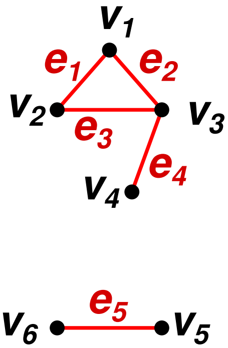
Zusammenhang
Ungerichteter Graph

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



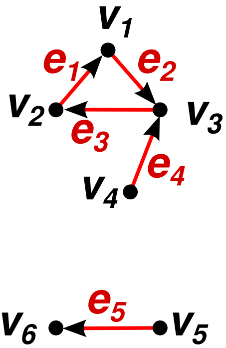
Zusammenhang
Ungerichteter Graph

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



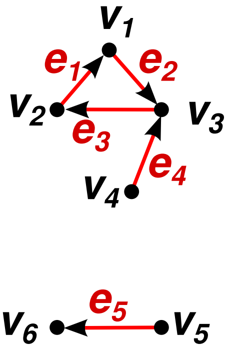
Zusammenhang
Gerichteter Graph

- Starker Zusammenhang von u und v zusammen, $:\Leftrightarrow$
  Es gibt gerichteten Weg von u nach v und von v nach u



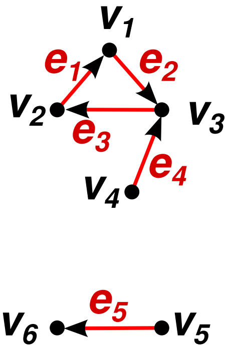
Zusammenhang
Gerichteter Graph

- 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



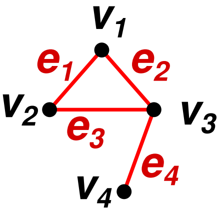
Zusammenhang
Gerichteter Graph

- 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



Datenstrukturen
Adjazenzmatrix für Ungerichtete Graphen

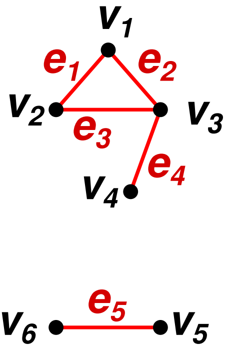
- Adjazenzmatrix A_G von $G(V, E)$



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

Datenstrukturen
Adjazenzmatrix für Ungerichtete Graphen

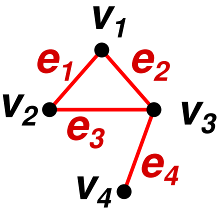
- Adjazenzmatrix A_G von $G(V, E)$
  - $n \times n$ Matrix mit $n = |V|$



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

Datenstrukturen
Adjazenzmatrix für Ungerichtete Graphen

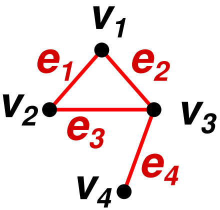
- Adjazenzmatrix A_G von $G(V, E)$
  - $n \times n$ Matrix mit $n = |V|$
  - $A_{ij} = 1$ falls $\{v_i, v_j\} \in E$, sonst $= 0$



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

Adjazenzmatrix für Ungerichtete Graphen

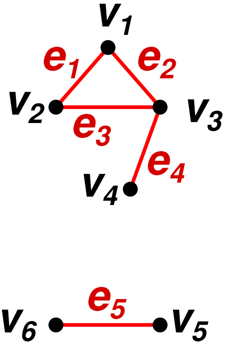
- Adjazenzmatrix A_G von $G(V, E)$
  - $n \times n$ Matrix mit $n = |V|$
  - $A_{ij} = 1$ falls $\{v_i, v_j\} \in E$, sonst $= 0$
  - Symmetrische Matrix



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

Datenstrukturen
Adjazenzmatrix für Ungerichtete Graphen

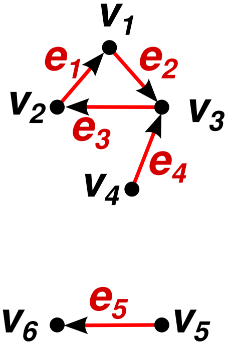
- Adjazenzmatrix A_G von $G(V, E)$
  - $n \times n$ Matrix mit $n = |V|$
  - $A_{ij} = 1$ falls $\{v_i, v_j\} \in E$, sonst $= 0$
  - Symmetrische Matrix
  - Statt 0 und 1 auch Gewichte möglich



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

Datenstrukturen
Adjazenzmatrix für Gerichtete Graphen

- 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 symmetrisch
  - 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} : $\mathcal{O}(1)$

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

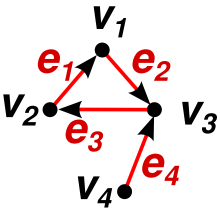
Datenstrukturen
Operationen auf Adjazenzmatrizen

TECHNISCHE UNIVERSITÄT DARMSTADT

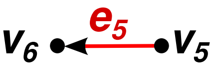
- Test, ob $(v_i, v_j) \in E$
  - Nachsehen in A_{ij} : $\mathcal{O}(1)$
- Welche v sind direkt mit u_i verbunden?
  - Zeile i durchgehen: $\mathcal{O}(n)$
  - Ineffizient bei vielen Nullen
- Größenveränderung nur schwer möglich

Datenstrukturen
(Knoten-Kanten-) Inzidenzmatrix

- Adjazenzmatrix A_G von $G(V, E)$
  - $m \times n$ Matrix mit $n = |V|, m = |E|$



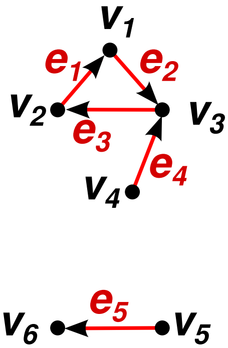
$$A_G = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$



Datenstrukturen
(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

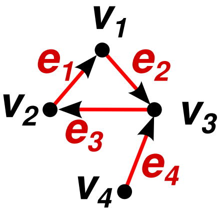


$$A_G = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

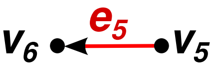
Datenstrukturen
(Knoten-Kanten-) Inzidenzmatrix

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

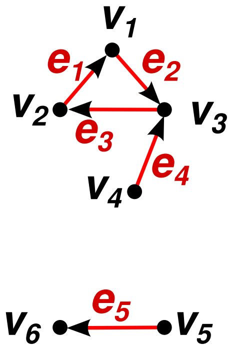


$$A_G = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$



Datenstrukturen
(Knoten-Kanten-) Inzidenzmatrix

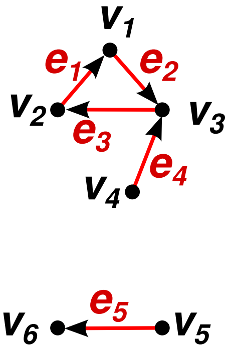
- Adjazenzmatrix A_G von $G(V, E)$
  - $m \times n$ Matrix mit $n = |V|$, $m = |E|$
  - Zeilen entsprechen Kanten, Spalten Knoten
  - Genau zwei nicht 0-Einträge pro Zeile
  - Bei gerichteten Graphen:
    $e_m = \{v_i, v_j\} \in E$, $A_{mj} = 1$, $A_{mi} = -1$



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

Datenstrukturen
(Knoten-Kanten-) Inzidenzmatrix

- 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

Datenstrukturen
Adjazenzliste

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

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: $\mathcal{O}(k)$

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

- Aufgabe
  - *Besuche alle V und E von $G(V, E)$!*
  - Jedes Element genau einmal!

Graphen Travesierung

- Aufgabe
  - *Besuche alle V und E von $G(V, E)$!*
  - Jedes Element genau einmal!
- Unterschiedliche Reihenfolgen möglich

Graphen Travesierung

- Aufgabe
  - *Besuche alle V und E von $G(V, E)$!*
  - Jedes Element genau einmal!
- Unterschiedliche Reihenfolgen möglich
- Weit verbreitet

Graphen Travesierung

- Aufgabe
  - *Besuche alle* V *und* E *von* $G(V, E)$*!*
  - Jedes Element genau einmal!
- Unterschiedliche Reihenfolgen möglich
- Weit verbreitet

  Tiefensuche Suche von Ursprungsknoten entfernen

Graphen Travesierung

- 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

dfs(vertex v)
begin
  v.mark := 0;
  v.process();
  foreach *(v,u)* $\in$ *E* **do**
    (v,u).process();
    if *(u.mark)* **then** dfs(u);
    ;

main()
begin
  foreach $v \in V$ **do**
    v.mark := 1;
  foreach $v \in V$ **do**
    if *(v.mark)* **then** dfs(v);
    ;

Graphen Travesierung
Tiefensuche (DFS) – Theoretisch

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

Graphen Travesierung
Tiefensuche (DFS) – Theoretisch

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

  $\Rightarrow \mathcal{O}(|V| + |E|)$

Graphen Travesierung
Tiefensuche (DFS) – Theoretisch

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

  $\Rightarrow \mathcal{O}(|V| + |E|)$
- Anwendungsbeispiele

Graphen Travesierung
Tiefensuche (DFS) – Theoretisch

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

  $\Rightarrow \mathcal{O}(|V| + |E|)$
- Anwendungsbeispiele
  - Systematischer Graphdurchlauf

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);
    until Q == ();
```

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

Graphen Travesierung
Breitensuche (BFS) – Theoretisch

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

Graphen Travesierung
Breitensuche (BFS) – Theoretisch

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

  $\Rightarrow$ $\mathcal{O}(|V| + |E|)$

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

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

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

Graphen Travesierung
Breitensuche (BFS) – Theoretisch

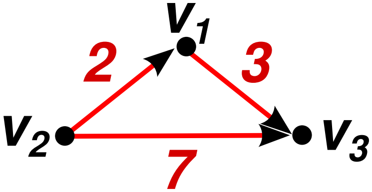
TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

Kürzester Pfad

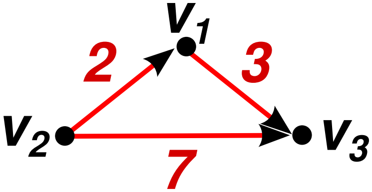
- Aufgabe: Bestimme den kürzesten Pfad vom Startknoten zu Zielknoten



Kürzester Pfad

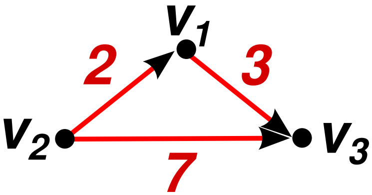
TECHNISCHE UNIVERSITÄT DARMSTADT

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



Kürzester Pfad

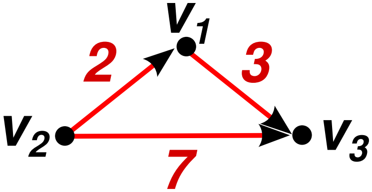
- Aufgabe: Bestimme den kürzesten Pfad vom Startknoten zu Zielknoten
  - Manchmal auch: zu allen anderen Knoten
- Bei ungewichteten Graphen z.B. mit BFS
  - Erweitert um Verwaltung der Pfade



Kürzester Pfad

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aufgabe: 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
- BFS nicht bei gewichteten Graphen!
  - Niedrige Anzahl von Kanten nicht immer kürzester (leichtester) Weg



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**

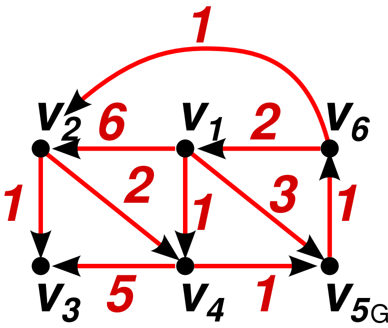
  if $((v_s, u) \in E)$ **then**

    `u.dist := (vs, u).weight;`

  else `u.dist := +∞;`

  ;

⋮



Kürzester Pfad von v_1 nach v_3

Embedded Systems & Applications

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

⋮

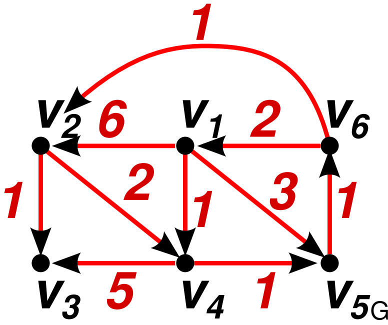
while $(v_t \notin T)$ **do**

```
u := V.findmin(dist);
T := T ∪ {u};
V := V ∪ {u};
```

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

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

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



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 - {vs };
T := { vs };
vs.dist := 0;
```

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

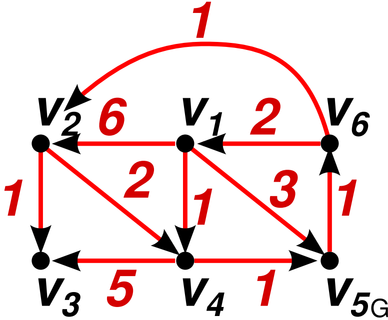
    if $((v_s, u) \in E)$ **then**

        `u.dist := (vs, u).weight;`

    else `u.dist := +∞;`

    ;

⋮



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)

⋮

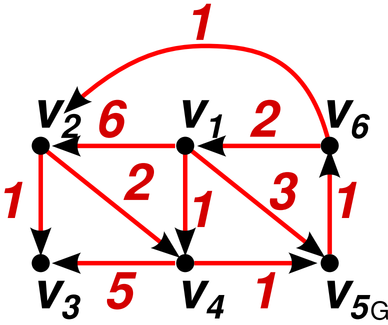
while $(v_t \notin T)$ **do**

```
u := V.findmin(dist);
T := T ∪ {u};
V := V ∪ {u};
```

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

if *(v.dist > u.dist + (u, v).weight)* **then**

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



Kürzester Pfad von v_1 nach v_3

T= $\{v_1\}$

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

Kürzester Pfad
Dijkstra – Praktisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

⋮

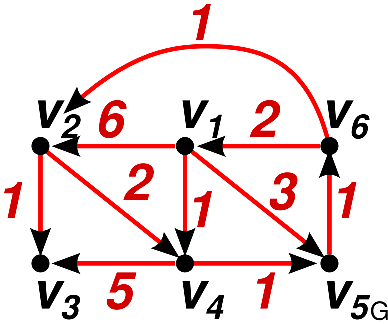
while $(v_t \notin T)$ **do**

```
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
```

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

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

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



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

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

⋮

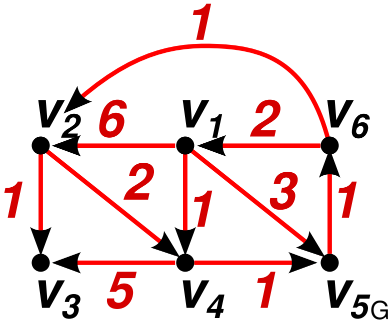
while $(v_t \notin T)$ **do**

```
u := V.findmin(dist);
T := T ∪ {u};
V := V ∪ {u};
```

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

if *(v.dist > u.dist + (u, v).weight)* **then**

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



Kürzester Pfad von v_1 nach v_3

T= $\{v_1, v_4\}$

$v_i.dist =$ 0 6 6 1 2 ∞

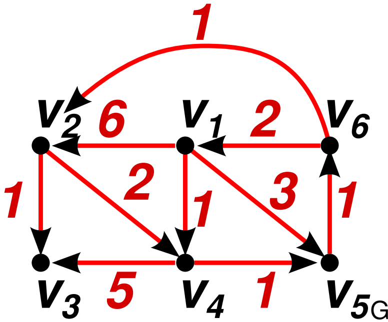
Kürzester Pfad
Dijkstra – Praktisch

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

⋮

while $(v_t \notin T)$ **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;
```



Kürzester Pfad von v_1 nach v_3

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

$v_i.dist = \quad 0 \quad 6 \quad 6 \quad 1 \quad 2 \quad \infty$

Kürzester Pfad
Dijkstra – Praktisch

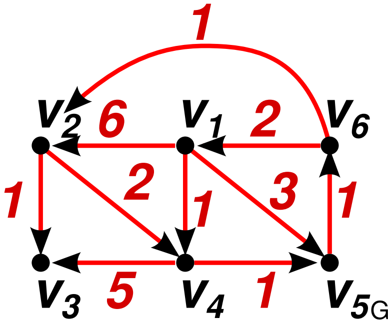
TECHNISCHE UNIVERSITÄT DARMSTADT

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

⋮

while $(v_t \notin T)$ **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;
```



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)

⋮

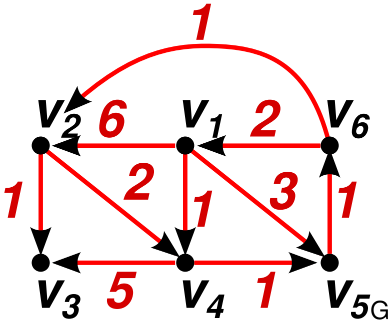
while $(v_t \notin T)$ **do**

```
u := V.findmin(dist);
T := T ∪ {u};
V := V ∪ {u};
```

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

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

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



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

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

⋮

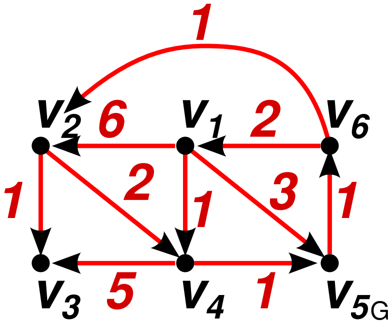
while $(v_t \notin T)$ **do**

```
u := V.findmin(dist);
T := T ∪ {u};
V := V ∪ {u};
```

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

if *(v.dist > u.dist + (u, v).weight)* **then**

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



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)

⋮

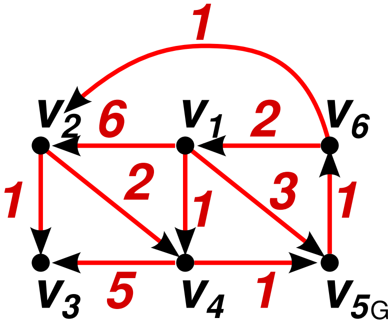
while $(v_t \notin T)$ **do**

```
    u := V.findmin(dist);
    T := T ∪ {u};
    V := V ∪ {u};
```

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

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

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



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

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

⋮

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

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

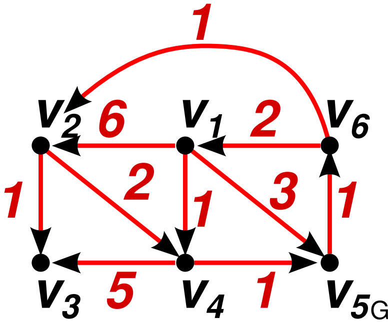
  `T := T` $\cup$ `{u};`

  `V := V` $\cup$ `{u};`

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

    if *(v.dist > u.dist + (u, v).weight)* **then**

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



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

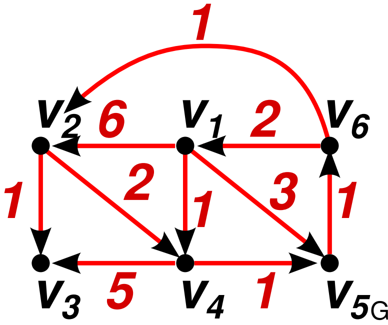
Kürzester Pfad
Dijkstra – Praktisch

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

⋮

while $(v_t \notin T)$ **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;
```



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

Kürzester Pfad
Dijkstra – Theoretisch

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

    $\Rightarrow$ $\mathcal{O}(|V|^2)$

Kürzester Pfad
Dijkstra – Theoretisch

- 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

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

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

    $\Rightarrow \mathcal{O}(|V|^2)$

Dijkstra – Theoretisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

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

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

Kürzester Pfad
Dijkstra – Theoretisch

TECHNISCHE UNIVERSITÄT DARMSTADT

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

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

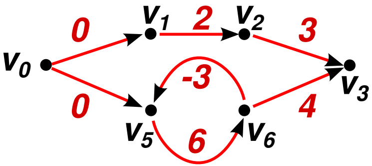
    $\Rightarrow$ $\mathcal{O}(|V|^2)$
  - Gesamtaufwand $\mathcal{O}(|V|^2 + |V|^2) = \mathcal{O}(|V|^2)$
- Wichtig: Funktioniert nur bei Kantengewichten ≥ 0!

Längster Pfad

Algorithmus abhängig vom Graphen:

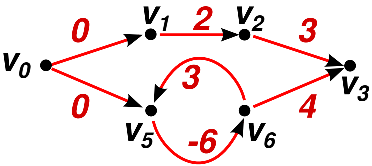
Zyklenfreier Graph: OK, ähnlich zu BFS

Mit positivem Zyklus



Undefiniert!

Mit negativem Zyklus



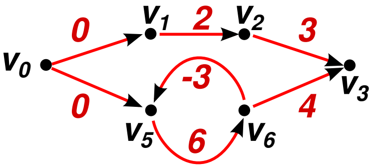
OK

Algorithmus abhängig vom Graphen:

Zyklenfreier Graph: OK, ähnlich zu BFS

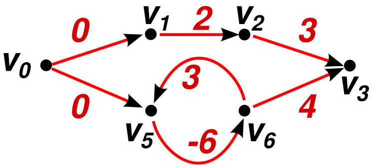
Graph mit Zyklen: Unterscheidung nach Zyklusart

Mit positivem Zyklus



Undefiniert!

Mit negativem Zyklus



OK

Längster Weg
Zyklenfreie Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

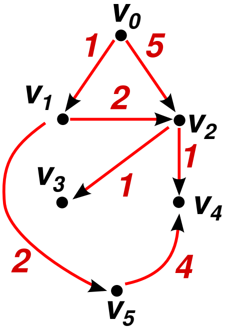
```
main()
begin
    foreach 0 ≤ i < n do
        x_i := 0
    longestPath(G) ;
```

Vorraussetzung:

Graph ist ein DAG
(Directed Acyclic Graph)!

```
longestPath(G):
begin
    foreach v_i in V do
        p_i := v_i.inDegree()
    Set Q := {v_0};
    while (Q ≠ ∅) do
        v_i := Q.pickany();
        Q := Q \ {v_i} ;
        foreach (v_i, v_j) ∈ E do
            x_j := max(x_j, x_i + d_ij);
            p_j := p_j - 1;
            if p_j ≤ 0 then
                Q := Q ∪ {v_j}
```

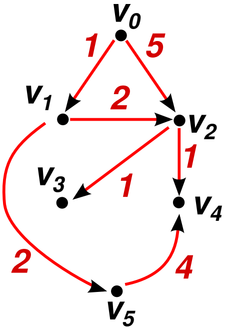
Längster Pfad
DAG Beispiel



Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0

Längster Pfad
DAG Beispiel

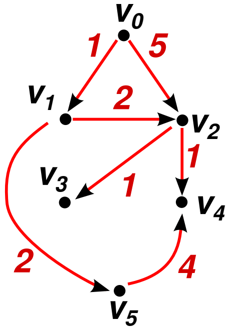
TECHNISCHE UNIVERSITÄT DARMSTADT



Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0

Längster Pfad
DAG Beispiel

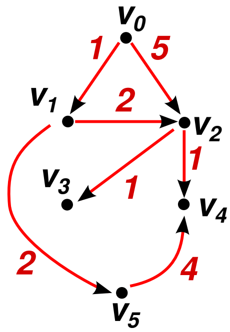
TECHNISCHE UNIVERSITÄT DARMSTADT



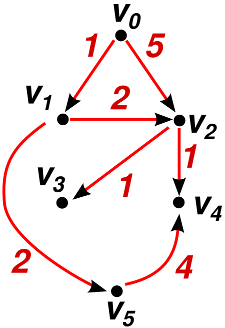
Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0
$\{v_1\}$	0	0	1	2	0	1	5	0	0	3

Längster Pfad
DAG Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



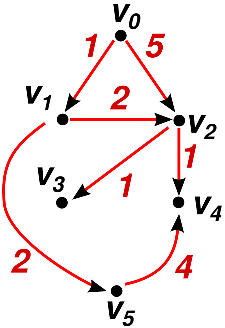
Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0
$\{v_1\}$	0	0	1	2	0	1	5	0	0	3
$\{v_2, v_5\}$	0	0	0	1	0	1	5	6	6	3



Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0
$\{v_1\}$	0	0	1	2	0	1	5	0	0	3
$\{v_2, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_3, v_5\}$	0	0	0	1	0	1	5	6	6	3

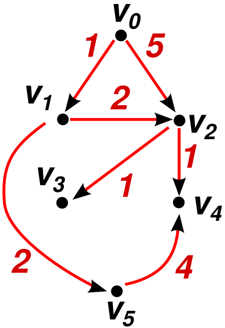
Längster Pfad
DAG Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0
$\{v_1\}$	0	0	1	2	0	1	5	0	0	3
$\{v_2, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_3, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_5\}$	0	0	0	0	0	1	5	6	7	3

Längster Pfad
DAG Beispiel



Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0
$\{v_1\}$	0	0	1	2	0	1	5	0	0	3
$\{v_2, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_3, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_5\}$	0	0	0	0	0	1	5	6	7	3
$\{v_4\}$	0	0	0	0	0	1	5	6	7	3

Graphen mit Zyklen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nur mit *negativen* Zyklen

Graphen mit Zyklen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nur mit *negativen* Zyklen
- Erkenne positive Zyklen

Graphen mit Zyklen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nur mit *negativen* Zyklen
- Erkenne positive Zyklen
- Aber lokalisere sie nicht

```
foreach 0 ≤ i < n do
    x_i := −∞
x_0 := 0 ; loop_count = 0 ;
repeat
    is_modified := false ;
    longestPath(G_f) ;
    foreach (v_i, v_j) ∈ E_b do
        if x_j < (x_i + d_ij) then
            x_j := x_i + d_ij ;
            is_modified := true ;
    if ++loop_count > |E_b| && is_modified) then
        error(positive cycle!");
until ! is_modified;
```

Idee: Zyklen auftrennen

Längster Pfad
Liao-Wong

TECHNISCHE UNIVERSITÄT DARMSTADT

foreach $0 \leq i < n$ **do**
  $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
repeat
  is_modified := false ;
  longestPath(G_f) ;
  foreach $(v_i, v_j) \in E_b$ **do**
    if $x_j < (x_i + d_{ij})$ **then**
      $x_j := x_i + d_{ij}$;
      is_modified := true ;
  if $++$ *loop_count* $> |E_b|$ *&& is_modified)* **then**
    error(positive cycle!");
until *! is_modified*;

Idee: Zyklen auftrennen

- Kanten E_f: min. Distanz

Längster Pfad
Liao-Wong

TECHNISCHE UNIVERSITÄT DARMSTADT

```
foreach 0 ≤ i < n do
  └ x_i := −∞
x_0 := 0 ; loop_count = 0 ;
repeat
  | is_modified := false ;
  | longestPath(G_f) ;
  | foreach (v_i, v_j) ∈ E_b do
  |   | if x_j < (x_i + d_ij) then
  |   |   | x_j := x_i + d_ij ;
  |   |   └ is_modified := true ;
  |   └
  | if ++loop_count > |E_b| && is_modified) then
  |   └ error(positive cycle!");
until ! is_modified;
```

Idee: Zyklen auftrennen

- Kanten E_f: min. Distanz
- Kanten E_b: max. Distanz

Längster Pfad
Liao-Wong

```
foreach 0 ≤ i < n do
  xi := −∞
x0 := 0 ; loop_count = 0 ;
repeat
    is_modified := false ;
    longestPath(Gf) ;
    foreach (vi, vj) ∈ Eb do
        if xj < (xi + dij) then
            xj := xi + dij ;
            is_modified := true ;
    if ++loop_count > |Eb| && is_modified) then
        error(positive cycle!");
until !is_modified;
```

Idee: Zyklen auftrennen

- Kanten E_f: min. Distanz
- Kanten E_b: max. Distanz

$\Rightarrow$ Teilgraph $G_f(V, E_f)$

Längster Pfad
Liao-Wong

TECHNISCHE UNIVERSITÄT DARMSTADT

```
foreach 0 ≤ i < n do
  x_i := −∞
x_0 := 0 ; loop_count = 0 ;
repeat
  is_modified := false ;
  longestPath(G_f) ;
  foreach (v_i, v_j) ∈ E_b do
    if x_j < (x_i + d_ij) then
      x_j := x_i + d_ij ;
      is_modified := true ;
  if ++loop_count > |E_b| && is_modified) then
    error(positive cycle!");
until !is_modified;
```

Idee: Zyklen auftrennen

- Kanten E_f: min. Distanz
- Kanten E_b: max. Distanz

$\Rightarrow$ Teilgraph $G_f(V, E_f)$

- Löse LongestPath(G_f)

Embedded Systems & Applications

Längster Pfad
Liao-Wong

TECHNISCHE UNIVERSITÄT DARMSTADT

```
foreach 0 ≤ i < n do
  xi := −∞
x0 := 0 ; loop_count = 0 ;
repeat
    is_modified := false ;
    longestPath(Gf) ;
    foreach (vi, vj) ∈ Eb do
        if xj < (xi + dij) then
            xj := xi + dij ;
            is_modified := true ;
    if ++loop_count > |Eb| && is_modified) then
        error(positive cycle!");
until ! is_modified;
```

Idee: Zyklen auftrennen

- ▶ Kanten E_f: min. Distanz
- ▶ Kanten E_b: max. Distanz
- ⇒ Teilgraph $G_f(V, E_f)$
- ▶ Löse LongestPath(G_f)
- ▶ Korrigiere für entfernte E_b (Zyklen schließen)

Längster Pfad
Liao-Wong

foreach $0 \leq i < n$ **do**
    $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
repeat
    is_modified := false ;
    longestPath(G_f) ;
    foreach $(v_i, v_j) \in E_b$ **do**
        if $x_j < (x_i + d_{ij})$ **then**
            $x_j := x_i + d_{ij}$;
            is_modified := true ;
    if $++$ *loop_count* $> |E_b|$ *&& is_modified)* **then**
        error(positive cycle!");
until *! is_modified*;

Idee: Zyklen auftrennen

- Kanten E_f: min. Distanz
- Kanten E_b: max. Distanz

$\Rightarrow$ Teilgraph $G_f(V, E_f)$

- Löse LongestPath(G_f)
- Korrigiere für entfernte E_b (Zyklen schließen)
- Jedes $e_b \in E_b$ max. $1\times$ im Pfad
  $\Rightarrow$ stabilisert sich in $|E_b|$

Längster Pfad
Liao-Wong

```
foreach 0 ≤ i < n do
  x_i := −∞
x_0 := 0 ; loop_count = 0 ;
repeat
    is_modified := false ;
    longestPath(G_f) ;
    foreach (v_i, v_j) ∈ E_b do
        if x_j < (x_i + d_ij) then
            x_j := x_i + d_ij ;
            is_modified := true ;
    if ++loop_count > |E_b| && is_modified) then
        error(positive cycle!");
until ! is_modified;
```

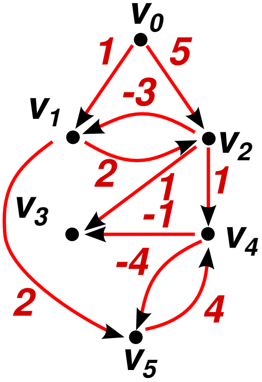
Idee: Zyklen auftrennen

- Kanten E_f: min. Distanz
- Kanten E_b: max. Distanz

$\Rightarrow$ Teilgraph $G_f(V, E_f)$

- Löse LongestPath(G_f)
- Korrigiere für entfernte E_b (Zyklen schließen)
- Jedes $e_b \in E_b$ max. $1\times$ im Pfad
  $\Rightarrow$ stabilisert sich in $|E_b|$
- Wenn nicht
  $\Rightarrow$ überbeschränkt

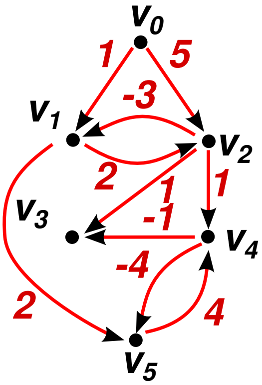
Längster Pfad
Liao-Wong Beispiel



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

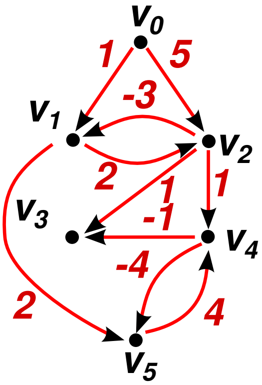
Längster Pfad
Liao-Wong Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3

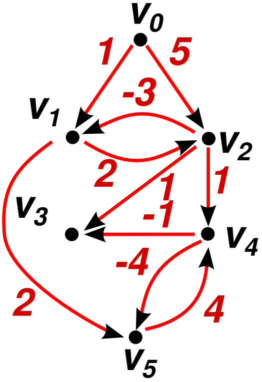
Längster Pfad
Liao-Wong Beispiel



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3

Längster Pfad
Liao-Wong Beispiel

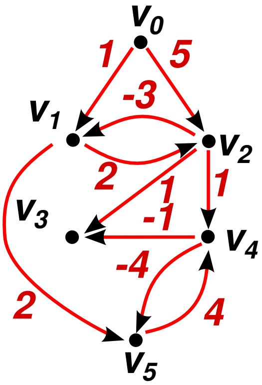
TECHNISCHE UNIVERSITÄT DARMSTADT



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3
Vor 2	2	5	6	8	4

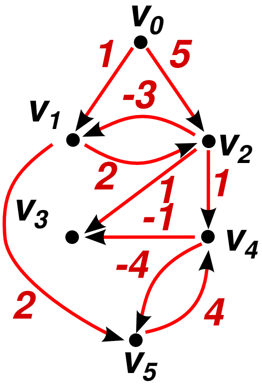
Längster Pfad
Liao-Wong Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3
Vor 2	2	5	6	8	4
Zurück 2	2	5	7	8	4

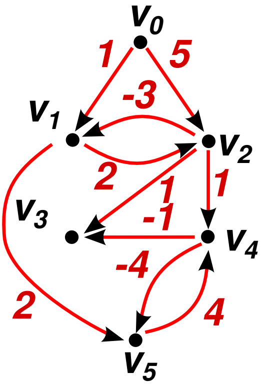
Längster Pfad
Liao-Wong Beispiel



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3
Vor 2	2	5	6	8	4
Zurück 2	2	5	7	8	4
Vor 3	2	5	7	8	4

Längster Pfad
Liao-Wong Beispiel

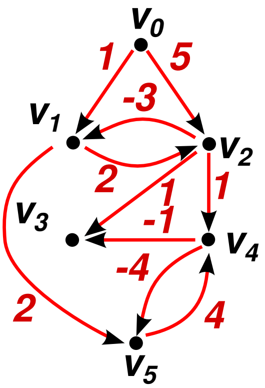
TECHNISCHE UNIVERSITÄT DARMSTADT



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3
Vor 2	2	5	6	8	4
Zurück 2	2	5	7	8	4
Vor 3	2	5	7	8	4
Zurück 3	2	5	7	8	4

Längster Pfad
Liao-Wong Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

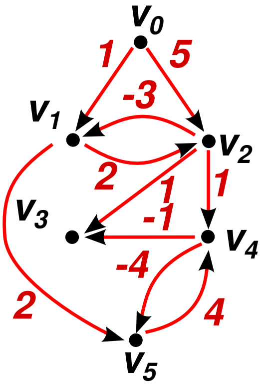


Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3
Vor 2	2	5	6	8	4
Zurück 2	2	5	7	8	4
Vor 3	2	5	7	8	4
Zurück 3	2	5	7	8	4

- Verbesserung: longestPath(G_f) bemerkt Änderung

Längster Pfad
Liao-Wong Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3
Vor 2	2	5	6	8	4
Zurück 2	2	5	7	8	4
Vor 3	2	5	7	8	4
Zurück 3	2	5	7	8	4

- Verbesserung: longestPath(G_f) bemerkt Änderung
- $\mathcal{O}(|E_f| \times |E_b|)$
  d.h. besonders gut, falls $|E_b| \ll |E_f|$

Längster Pfad
Bellman-Ford

TECHNISCHE UNIVERSITÄT DARMSTADT

foreach $0 \leq i < n$ **do**
    $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
$S_1 := \{v_0\}$; $S_2 := \emptyset$;
while $loop\_count \leq n \;\&\&\; S_1 \neq \emptyset$ **do**
    foreach $v_i \in S_1$ **do**
        foreach $(v_i, v_j) \in E$ **do**
            if $x_j < (x_i + d_{ij})$ **then**
                $x_j := x_i + d_{ij}$;
                $S_2 := S_2 \cup \{v_j\}$;
    $S_1 := S_2$; $S_2 := \emptyset$;
    ++ loop_count ;
if $loop\_count > n$ **then** error(positive cycle!");

Idee: Zwei Wellenfronten

Längster Pfad
Bellman-Ford

TECHNISCHE UNIVERSITÄT DARMSTADT

foreach $0 \leq i < n$ **do**
  $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
$S_1 := \{v_0\}$; $S_2 := \emptyset$;
while *loop_count* $\leq n$ && $S_1 \neq \emptyset$ **do**
  foreach $v_i \in S_1$ **do**
    foreach $(v_i, v_j) \in E$ **do**
      if $x_j < (x_i + d_{ij})$ **then**
        $x_j := x_i + d_{ij}$;
        $S_2 := S_2 \cup \{v_j\}$;
  $S_1 := S_2$; $S_2 := \emptyset$;
  ++ loop_count ;
if *loop_count* $> n$ **then** error(positive cycle!");

Idee: Zwei Wellenfronten

S_1 aktuelle

Längster Pfad
Bellman-Ford

TECHNISCHE UNIVERSITÄT DARMSTADT

foreach $0 \leq i < n$ **do**
    $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
$S_1 := \{v_0\}$; $S_2 := \emptyset$;
while *loop_count* $\leq n$ && $S_1 \neq \emptyset$ **do**
    foreach $v_i \in S_1$ **do**
        foreach $(v_i, v_j) \in E$ **do**
            if $x_j < (x_i + d_{ij})$ **then**
                $x_j := x_i + d_{ij}$;
                $S_2 := S_2 \cup \{v_j\}$;
    $S_1 := S_2$; $S_2 := \emptyset$;
    ++ loop_count ;
if *loop_count* $> n$ **then** error(positive cycle!");

Idee: Zwei Wellenfronten

- S_1 aktuelle
- S_2 nächste Iteration

Längster Pfad
Bellman-Ford

foreach $0 \leq i < n$ **do**
    $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
$S_1 := \{v_0\}$; $S_2 := \emptyset$;
while *loop_count* $\leq n$ && $S_1 \neq \emptyset$ **do**
    foreach $v_i \in S_1$ **do**
        foreach $(v_i, v_j) \in E$ **do**
            if $x_j < (x_i + d_{ij})$ **then**
                $x_j := x_i + d_{ij}$;
                $S_2 := S_2 \cup \{v_j\}$;
    $S_1 := S_2$; $S_2 := \emptyset$;
    ++ loop_count ;
if *loop_count* $> n$ **then** error(positive cycle!");

Idee: Zwei Wellenfronten

- S_1 aktuelle
- S_2 nächste Iteration
- Vergleichbar azyklischem LP
  aber mehrere Durchläufe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
foreach 0 ≤ i < n do
    x_i := −∞
x_0 := 0 ; loop_count = 0 ;
S_1 := {v_0} ; S_2 := ∅ ;
while loop_count ≤ n && S_1 ≠ ∅ do
    foreach v_i ∈ S_1 do
        foreach (v_i, v_j) ∈ E do
            if x_j < (x_i + d_ij) then
                x_j := x_i + d_ij ;
                S_2 := S_2 ∪ {v_j} ;
    S_1 := S_2 ; S_2 := ∅ ;
    ++ loop_count ;
if loop_count > n then  error(positive cycle!");
```

Idee: Zwei Wellenfronten

- S_1 aktuelle
- S_2 nächste Iteration
  - Vergleichbar azyklischem LP
    aber mehrere Durchläufe
  - In k-ter Iteration
    LP durch $k-1$ Knoten

Längster Pfad
Bellman-Ford

TECHNISCHE UNIVERSITÄT DARMSTADT

foreach $0 \leq i < n$ **do**
  $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
$S_1 := \{v_0\}$; $S_2 := \emptyset$;
while *loop_count* $\leq n$ && $S_1 \neq \emptyset$ **do**
  foreach $v_i \in S_1$ **do**
    foreach $(v_i, v_j) \in E$ **do**
      if $x_j < (x_i + d_{ij})$ **then**
        $x_j := x_i + d_{ij}$;
        $S_2 := S_2 \cup \{v_j\}$;
  $S_1 := S_2$; $S_2 := \emptyset$;
  ++ loop_count ;
if *loop_count* $> n$ **then** error(positive cycle!");

Idee: Zwei Wellenfronten

- S_1 aktuelle
- S_2 nächste Iteration
  - Vergleichbar azyklischem LP
    aber mehrere Durchläufe
  - In k-ter Iteration
    LP durch $k - 1$ Knoten
- $\Rightarrow$ Zyklendetektion
  LP $> n$ Knoten $\Rightarrow$ Zyklus!

Längster Pfad
Bellman-Ford

foreach $0 \leq i < n$ **do**
    $x_i := -\infty$

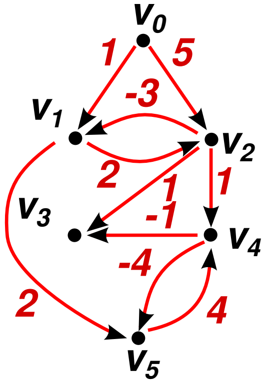
$x_0 := 0$; loop_count = 0 ;
$S_1 := \{v_0\}$; $S_2 := \emptyset$;
while *loop_count* $\leq n$ && $S_1 \neq \emptyset$ **do**
    foreach $v_i \in S_1$ **do**
        foreach $(v_i, v_j) \in E$ **do**
            if $x_j < (x_i + d_{ij})$ **then**
                $x_j := x_i + d_{ij}$;
                $S_2 := S_2 \cup \{v_j\}$;
    $S_1 := S_2$; $S_2 := \emptyset$;
    ++ loop_count ;

if *loop_count* $> n$ **then** error(positive cycle!");

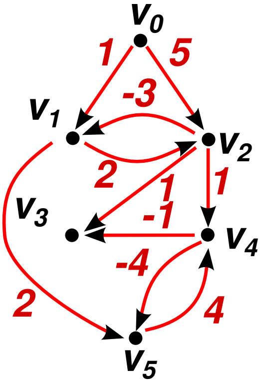
Idee: Zwei Wellenfronten

- S_1 aktuelle
- S_2 nächste Iteration
- Vergleichbar azyklischem LP
  aber mehrere Durchläufe
- In k-ter Iteration
  LP durch $k - 1$ Knoten
- $\Rightarrow$ Zyklendetektion
  LP $> n$ Knoten $\Rightarrow$ Zyklus!
- $\mathcal{O}(n^3)$, avg. $\mathcal{O}(n^{1.5})$

Längster Pfad
Bellman-Ford Beispiel



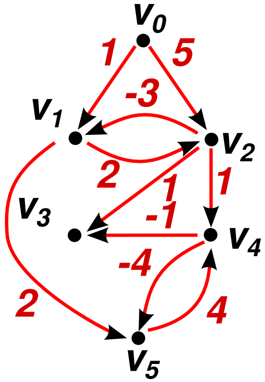
S_1	x_1	x_2	x_3	x_4	x_5



S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Längster Pfad
Bellman-Ford Beispiel

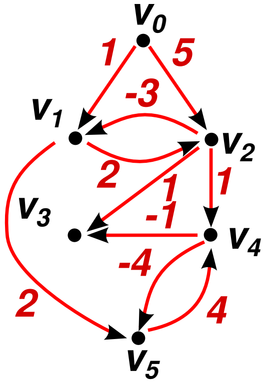
TECHNISCHE UNIVERSITÄT DARMSTADT



S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$

Längster Pfad
Bellman-Ford Beispiel

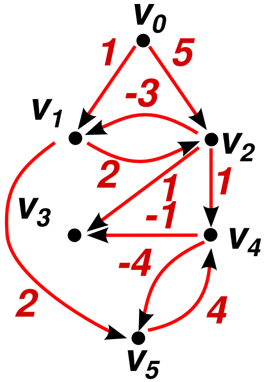
TECHNISCHE UNIVERSITÄT DARMSTADT



S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3

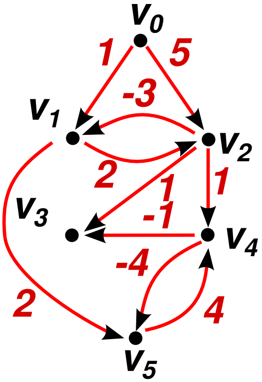
Längster Pfad
Bellman-Ford Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3
$\{v_1, v_3, v_4, v_5\}$	2	5	6	7	4

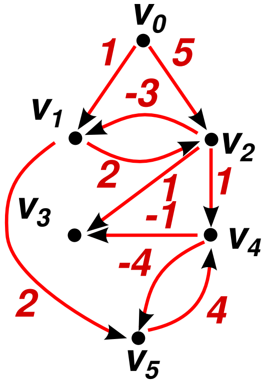
Längster Pfad
Bellman-Ford Beispiel



S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3
$\{v_1, v_3, v_4, v_5\}$	2	5	6	7	4
$\{v_4, v_5\}$	2	5	6	8	4

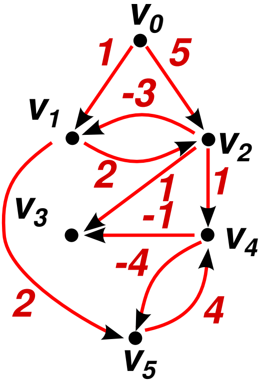
Längster Pfad
Bellman-Ford Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3
$\{v_1, v_3, v_4, v_5\}$	2	5	6	7	4
$\{v_4, v_5\}$	2	5	6	8	4
$\{v_4\}$	2	5	7	8	4

Längster Pfad
Bellman-Ford Beispiel



S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3
$\{v_1, v_3, v_4, v_5\}$	2	5	6	7	4
$\{v_4, v_5\}$	2	5	6	8	4
$\{v_4\}$	2	5	7	8	4
$\{v_3\}$	2	5	7	8	4

Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1

Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit

Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen

Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen
  - Alle Gewichte positiv
    $\Rightarrow$ SP in P, LP ist NP-vollständig

Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen
  - Alle Gewichte positiv
    $\Rightarrow$ SP in P, LP ist NP-vollständig
  - Alle Gewichte negativ
    $\Rightarrow$ LP in P, SP ist NP-vollständig

Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen
  - Alle Gewichte positiv
    $\Rightarrow$ SP in P, LP ist NP-vollständig
  - Alle Gewichte negativ
    $\Rightarrow$ LP in P, SP ist NP-vollständig
  - Keine positiven Zyklen: LP in P

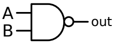
Pfadalgorithmen
Übersicht

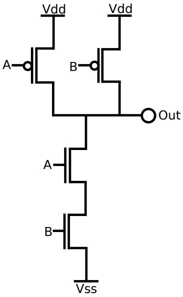
- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen
  - Alle Gewichte positiv
    $\Rightarrow$ SP in P, LP ist NP-vollständig
  - Alle Gewichte negativ
    $\Rightarrow$ LP in P, SP ist NP-vollständig
  - Keine positiven Zyklen: LP in P
  - Keine negativen Zyklen: SP in P

Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen
  - Alle Gewichte positiv
    $\Rightarrow$ SP in P, LP ist NP-vollständig
  - Alle Gewichte negativ
    $\Rightarrow$ LP in P, SP ist NP-vollständig
  - Keine positiven Zyklen: LP in P
  - Keine negativen Zyklen: SP in P
  - Sonst: NP-Vollständig

Sichten
Schematisch und Transistorlayout



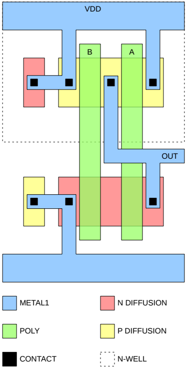


Bildquelle: Wikimedia Commons

Schematisches Schaltsymbol

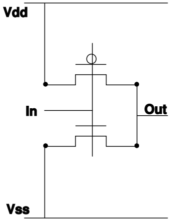
Transistorlayout

Sichten
Physikalisches/Geometrisches/Masken Layout

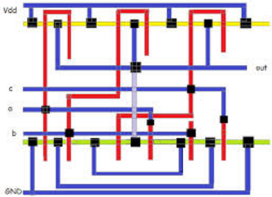


Bildquelle: Wikimedia Commons

Sichten
Symbolisches Layout



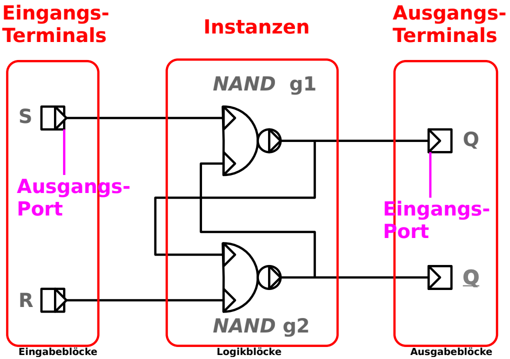
Symbolisches Layout



Stick-Diagramm

- Kein vollständiges Layout
- Keine absoluten geometrischen Angaben
- Nicht notwendige physikalische Angaben fehlen komplett (z.B. n- und p-Wells)
- *Symbole* für Elemente wie Transistoren oder Kontakte
- Länge, Breite, Layer noch variabel

Darstellungen von Schaltungen



Zelle und Master-Zelle

```
class cell_master {
   String name;
   truth_table func;
   Rect extent;
   set<port_master> ins, outs;
   ...
}

class cell {
   cell_master master;
   String name;
   set<port> ins, outs;
   ...
}
```

Port und Portmaster


```
class port_master {
   String name;
   Point location;
   ...
}

class port {
   port_master master;
   String id;
   cell parent;
   net connects;
   ...
}
```

Netz

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class net {
   String name;
   set<port> joined;
}
```

Schaltungsdarstellung
Objektzusammenfassung

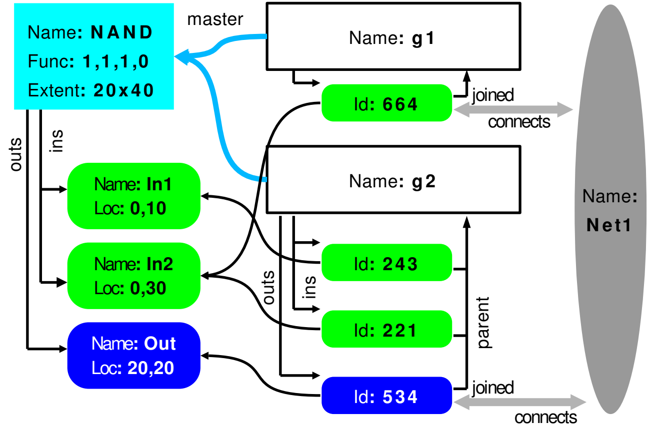
Instanz oder Zelle
- Ein Auftreten einer *Master-Zelle*
- Speichert instanzspezifische Eigenschaften

Master-Zelle Speichert Eigenschaften aller Instanzen

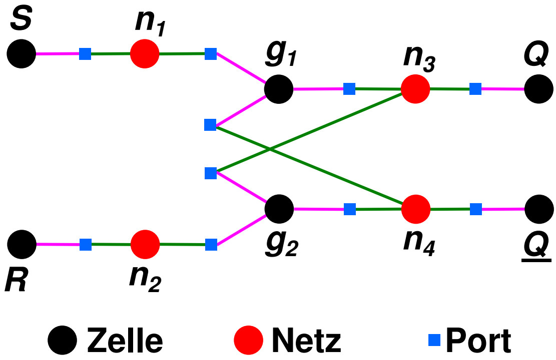
Netz Verbindung von mehreren Ports

Port
- Anschlusspunkt von Leitung an Zelle
- I.d.R nicht untereinander austauschbar
- Hierarchie: Terminals werden zu Ports

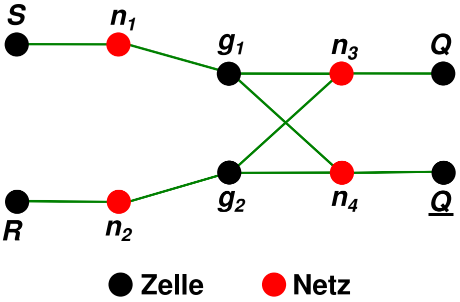
Schaltungsdarstellung
Beispiel



Schaltungsdarstellung – Graphmodellierung
Tripartiter Graph

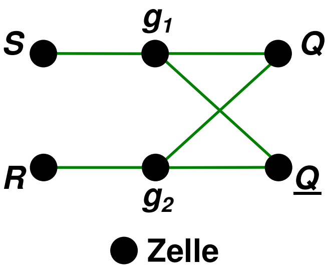


Schaltungsdarstellung – Graphmodellierung
Bipartiter Graph



- Weniger Details
- Verschmelzt Ports mit Zellen
- Äquivalent zu Hypergraph

Schaltungsdarstellung
Graphmodellierung



- Netze nicht mehr explizit modelliert
- Zellen an Netzen bilden jetzt Clique

Schaltungsdarstellungen
Übersicht

- Zelle-Port-Netz-Modell
- Tripartiter Graph
- Bipartiter Graph
- Clique-Modell

$\Downarrow$ ungenauer

Für Problem passendes Modell wählen

- Mehr Daten nicht immer besser

Konvertierungsroutinen bereitstellen

- Nur in ungenauere Darstellung möglich
- Buchführung über Herkunft von Daten

Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

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