

Vorlesungen

„Einführung in die Datenvisualisierung“

**„Angewandte Datenvisualisierung für
Medizinphysiker“**

Kapitel D

Visualisierung von Graphen

Dr. Frank Weichert

Informatik VII

Technische Universität Dortmund

<http://ls7-www.cs.uni-dortmund.de>

Nur zum persönlichen Gebrauch durch Mitglieder der Universität Dortmund

D.1. Graphentheorie

- D.1.1. Einleitung
- D.1.2. Gerichtete Graphen
- D.1.3. Ungerichtete Graphen
- D.1.4. Hypergraphen
- D.1.5. Speicherung
- D.1.6. Teilgraphen
- D.1.7. Pfade und Kreise
- D.1.8. Spezielle Eigenschaften

D.2. Zeichnen von Graphen

- D.2.1. Zeichenkonventionen
- D.2.2. Anforderungen
- D.2.3. Vorgehensweisen

D.3. Zeichenstrategien für Graphen

- D.3.1. Topologie-Form-Metrik-Ansatz
 - D.3.1.1. Verfeinerungsebenen
 - D.3.1.2. Algorithmus
 - D.3.1.3. Beispiel
- D.3.2. Hierarchischer Ansatz
 - D.3.2.1. Algorithmus
 - D.3.2.2. Beispiele
 - D.3.2.3. DAG
 - D.3.2.4. Bemerkungen
- D.3.3. Sichtbarkeitsansatz
 - D.3.3.1. Algorithmus
 - D.3.3.2. Beispiel
- D.3.4. Verfeinerungsansatz
 - D.3.4.1. Algorithmus
 - D.3.4.2. Beispiel

D.4. Schichtenweises Zeichnen allg. gerichteter Graphen

- D.4.1. Einleitung
- D.4.2. Algorithmus von Sugiyama
 - D.4.2.1. Übersicht
 - D.4.2.2. Entfernen von Zyklen (Schritt 1)
 - D.4.2.3. Schichtzuordnung (Schritt 2)
 - D.4.2.4. Kreuzungsreduktion (Schritt 3)
 - D.4.2.5. Waagrechte Koordinatenzuweisung (Schritt 4)
 - D.4.2.6. Bemerkungen

D.5. (Binäre) Bäume

- D.5.1. Konventionen
- D.5.2. Stufenweises Zeichnen
 - D.5.2.1. Einfacher Algorithmus
 - D.5.2.2. Algorithmus von Reingold/Tilford
- D.5.3. Horizontal-vertikal-Zeichnen
 - D.5.3.1. Algorithmus
 - D.5.3.2. Eigenschaften
- D.5.4. Zeichnen durch rekursives Drehen
 - D.5.4.1. Algorithmus
 - D.5.4.2. Eigenschaften
- D.5.5. Freie Bäume
- D.5.6. Bäume mit ausgedehnten Knoten

D.6. Vereinfachung von Polygonzügen

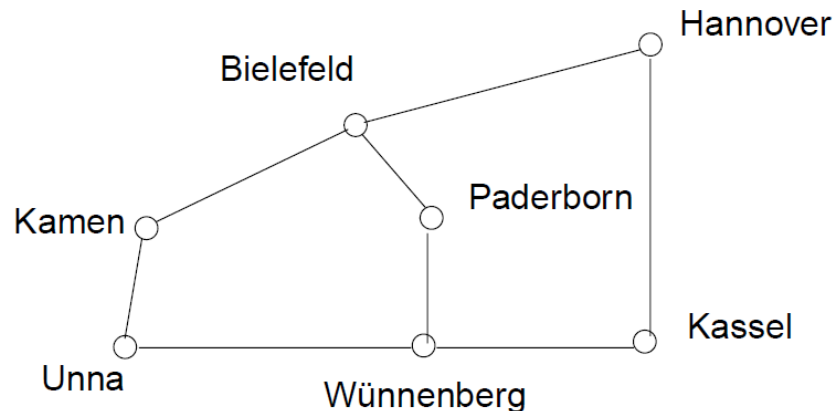
- D.6.1. Einführung
- D.6.2. min - # - Version (Imai, Iri)
- D.6.3. Fehlermaße
- D.6.4. min - e - Version
- D.6.5. Douglas-Peucker-Algorithmus

Zu visualisierende Entitäten (Objekte) besitzen häufig **inhärente Relationen**

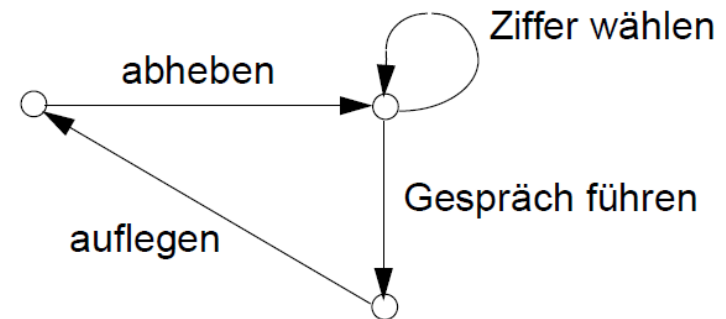
Ein Graph ist eine Abstraktion aus Knoten und Kanten zur Visualisierung von Objekten und Beziehungen (Relationen, Abhängigkeiten):

- **Knoten**: Eine Menge gleichartiger Objekte
- **Kanten**: Beziehung zwischen je zwei Objekten, 2-stellige Relation über Knoten

Unterschieden wird zwischen **ungerichteten** oder **gerichteten** Graphen



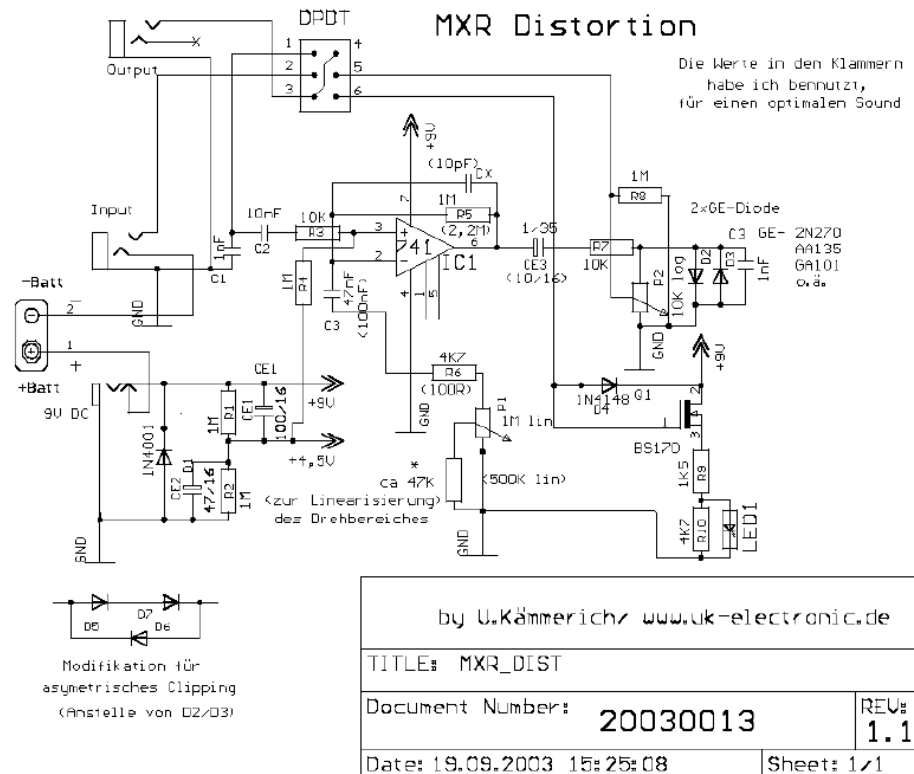
ungerichtet



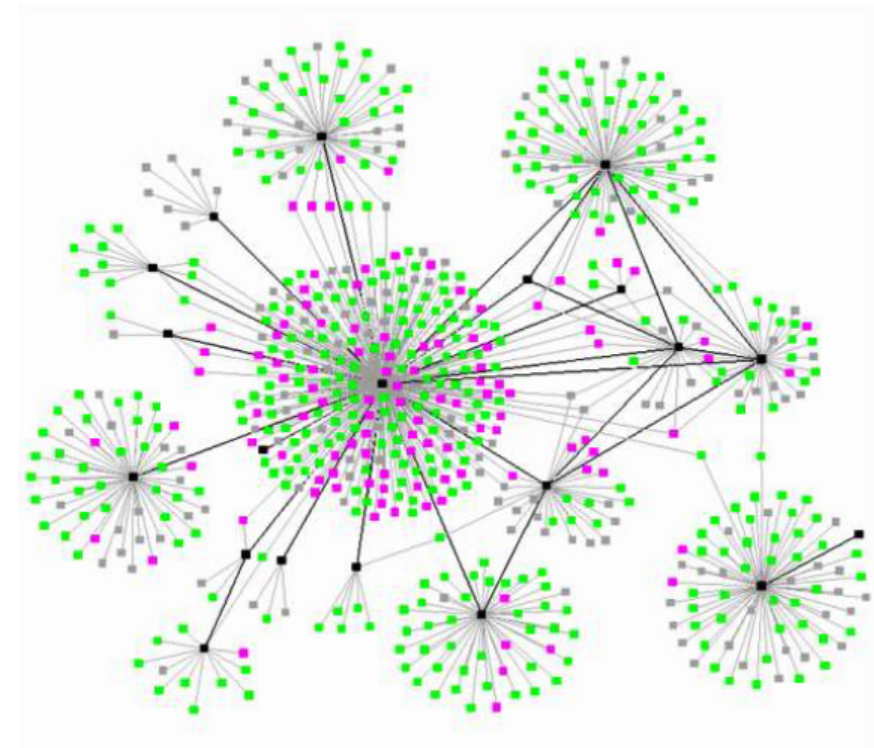
gerichtet

Typische Anwendungen

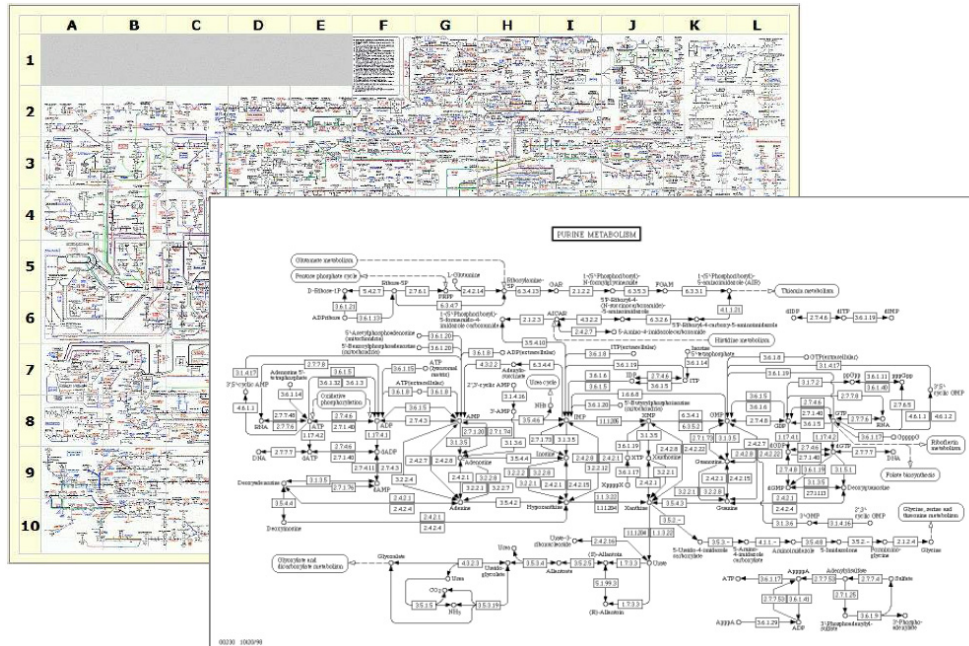
- Organigramm einer Firma
- Projektmanagement (z. B. Project Evaluation and Review Technique - PERT)
- Soziale Netzwerke
- Taxonomie biologischer Arten
- Biochemische Reaktionen, speziell Stoffwechselsysteme
- Abhängigkeiten bei Genexpression
- Ähnlichkeiten von Dokumenten
- Semantische Netzwerke und Wissensrepräsentationsdiagramme
- Webseiten und Weblinks
- Szenengraphen und virtuelle Realität
- Datenstrukturen, insbesondere bei Compilern
- Strukturen objektorientierte Systeme (Klassenhierarchie, Datenfluss)
- Realzeitsystem (Zustandsdiagramme)
- Schaltkreisentwurf (very large system integration VLSI)



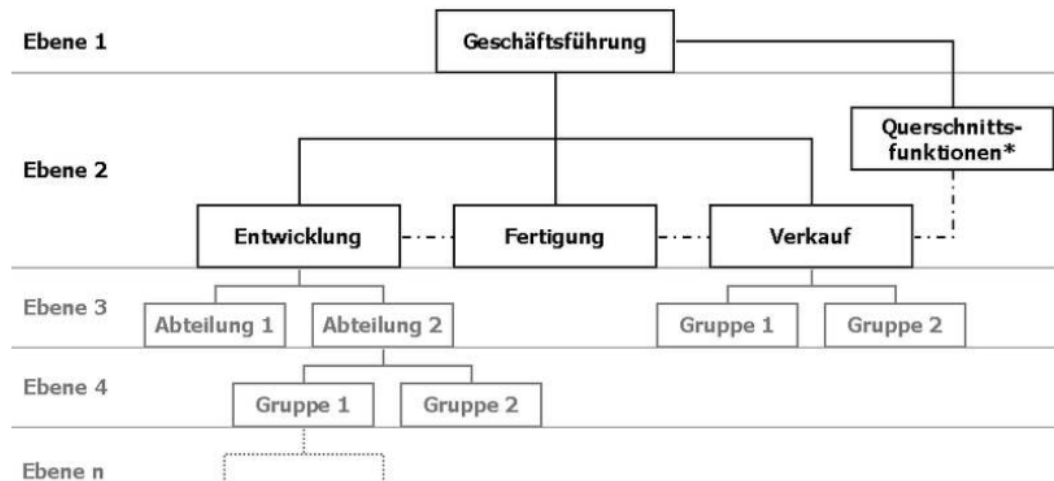
Schaltpläne



Soziale Netzwerke

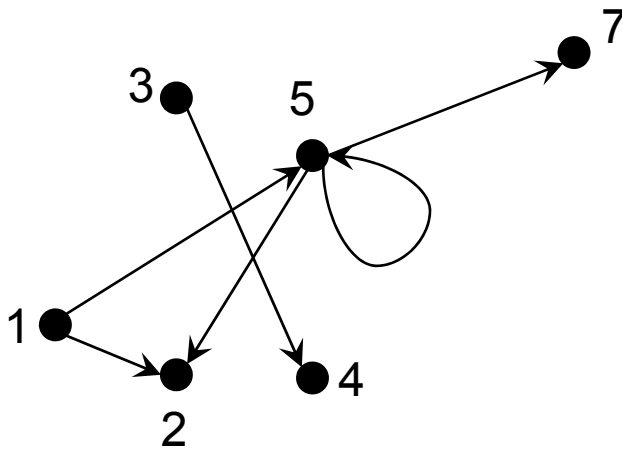


Metabolische Netzwerke



Organigramm einer Firma

- **Gerichteter Graph** oder **Digraph** ist ein Paar $G = (V, E)$ disjunkter Mengen mit $E \subseteq [V]^2$; die Elemente von E sind somit 2-elementige Teilmengen von V .
- Elemente von V repräsentieren die **Knoten** des Graphen G , Elemente E die **gerichteten Kanten** $\{v_i, v_j\}$ mit $v_i, v_j \in V$ (Kurznotation: $e_{i,j} \in E$).
- **Ingrad** von v_i ist die Anzahl eingehender Kanten, **Ausgrad** von v_i ist die Anzahl ausgehender Kanten. Knoten ohne ausgehende Kanten heißen **Senken**. Knoten ohne eingehende Kanten heißen **Quellen**.
- Eine Kante $e_{i,i}$ wird als **Schleife** oder Schlinge bezeichnet.

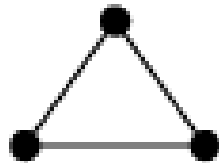
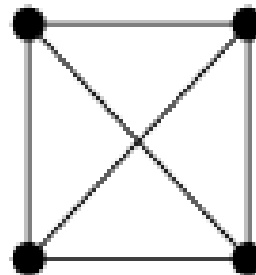
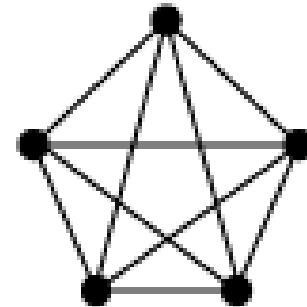


Graph $G = (V, E)$ mit

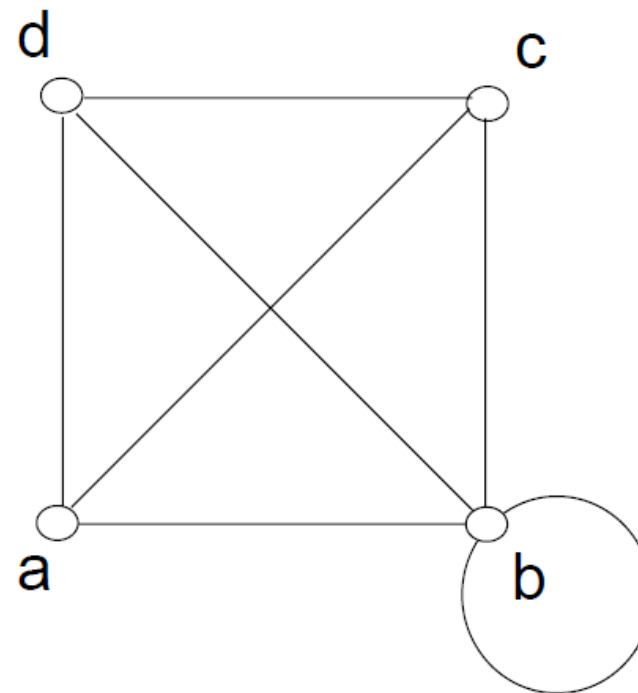
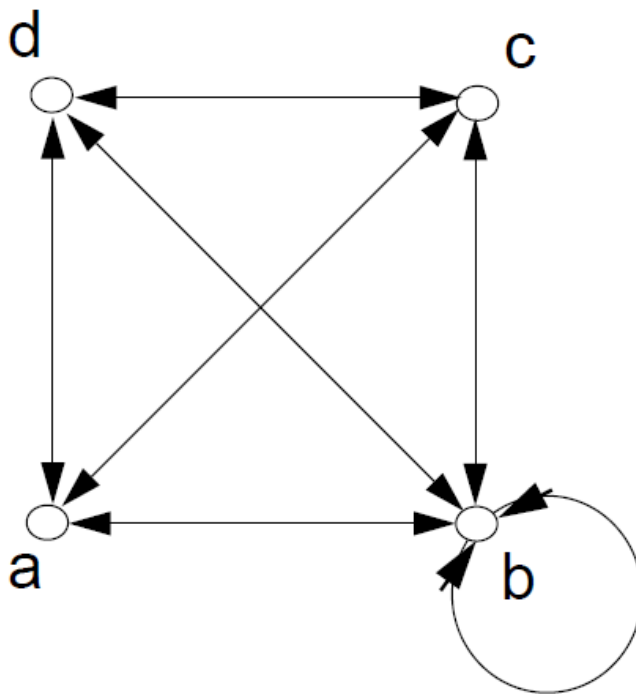
Knoten $V = \{1, \dots, 7\}$

Kanten $E = \{\{1,2\}, \{1,5\}, \{4,4\}, \{5,7\}, \{5,5\}\}$

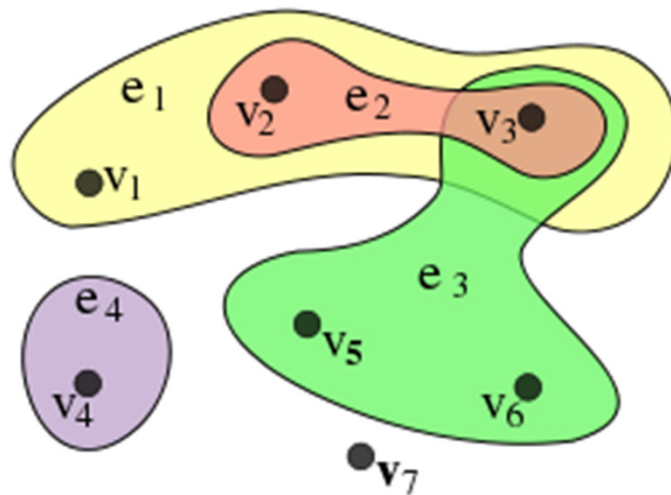
- Ein Knoten v und eine Kante e **inzidieren** miteinander und heißen inzident.
- Zwei Knoten v_i, v_j von G sind **adjazent** (benachbart) in G und heißen Nachbarn voneinander, wenn $e_{i,j} \in E(G)$ ist.
- Zwei Kanten $e_i \neq e_j$ sind adjazent, falls sie einen gemeinsamen Endknoten haben.
- Der **Grad** (oder die **Valenz**) $d_G(v) = d_v$ eines Knotens v ist die Anzahl $|E(v)|$ der mit v inzidenten Kanten; gemäß Definition somit die Anzahl der Nachbarn von v .
- Sind je zwei Ecken von G benachbart, so heißt G vollständig. Einen vollständigen Graphen auf n Ecken wird mit K^n bezeichnet; K^3 ist ein Dreieck.

 K^1  K^2  K^3  K^4  K^5

- **Ungerichteter Graph** beschreibt über die Kantenmenge eine symmetrische Relation
- In einem **ungerichteten Graphen** existiert zu jeder Kante $e_{i,j} \in E$ auch eine Kante $e_{j,i} \in E$



Ein **Hypergraph** ist ein Paar (V, E) disjunkter Mengen, bei dem jedes Element von E eine nicht leere Teilmenge (beliebiger Mächtigkeit) von V ist, d.h. jede Kante kann mehrere Anfangs- und Endknoten haben. Jeder Graph ist also ein spezieller Hypergraph.

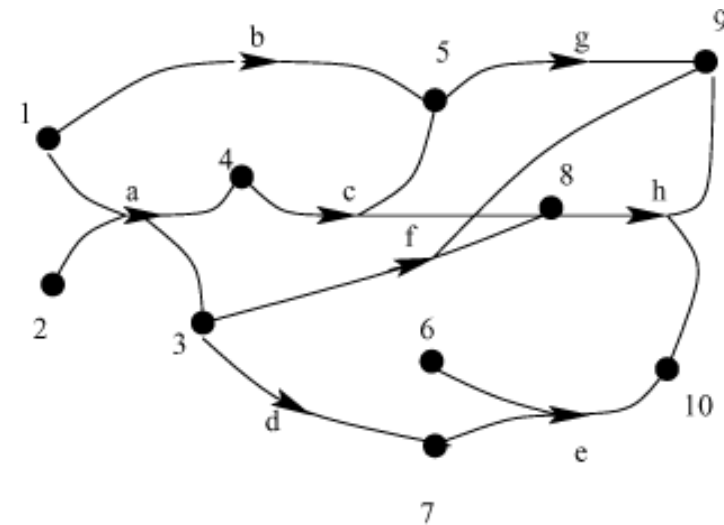


Hypergraph $G = (V, E)$

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$$

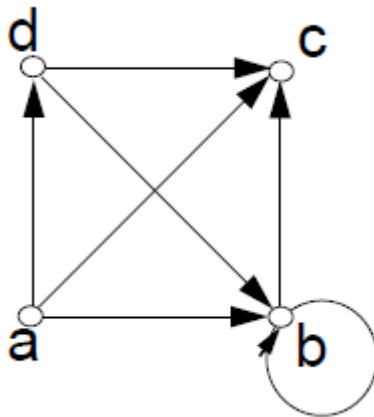
$$E = \{e_1, e_2, e_3, e_4\}$$

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



Gerichteter Hypergraph $G = (V, E)$

- Ein Graph $G = (V, E)$ mit n Knoten kann durch eine $n \times n$ **Adjazenzmatrix** A beschrieben werden, wobei $A_{i,j} = 1$, wenn $e_{i,j} \in E$ und $A_{i,j} = 0$ sonst.
Die Adjazenzmatrix eines (ungerichteten) Graphen ist immer symmetrisch.
- Adjazenzlisten** beschreiben zu jedem Knoten v_i eine Folge von Knoten, zu denen eine Kante $e_{i,j} \in E$



Adjazenzmatrix

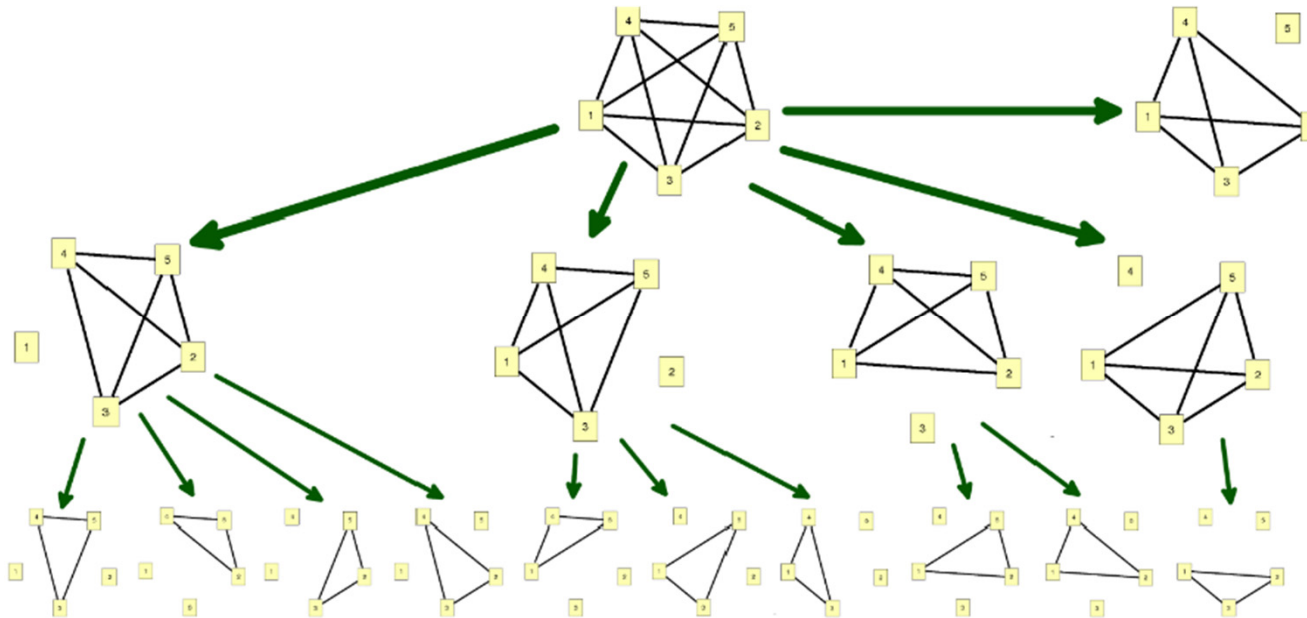
	a	b	c	d
a	0	1	1	1
b	0	1	1	0
c	0	0	0	0
d	0	1	1	0

Adjazenzlisten

a	(b, c, d)
b	(b, c)
c	()
d	(b, c)

Sei $G \cup G' = (V \cup V', E \cup E')$ und $G \cap G' = (V \cap V', E \cap E')$:

- Sofern $G \cap G' = \emptyset$, dann heißen G und G' **disjunkt**.
- Gilt $V \subseteq V'$ und $E \subseteq E'$, so ist G' ein **Teilgraph** von G und G ein **Obergraph** von G' , kurz $G \subseteq G'$. Informell bedeutet dies, dass der Graph G den Graphen G' enthält.
- Der Teilgraph G' heißt **induziert** (von V' in G), wenn er alle Kanten $e_{i,j} \in E$ mit $v_i, v_j \in V'$ enthält. Ein entsprechend induzierter Teilgraph ist ein Untergraph. Der Graph G' wird dann mit $G[V']$ bezeichnet.

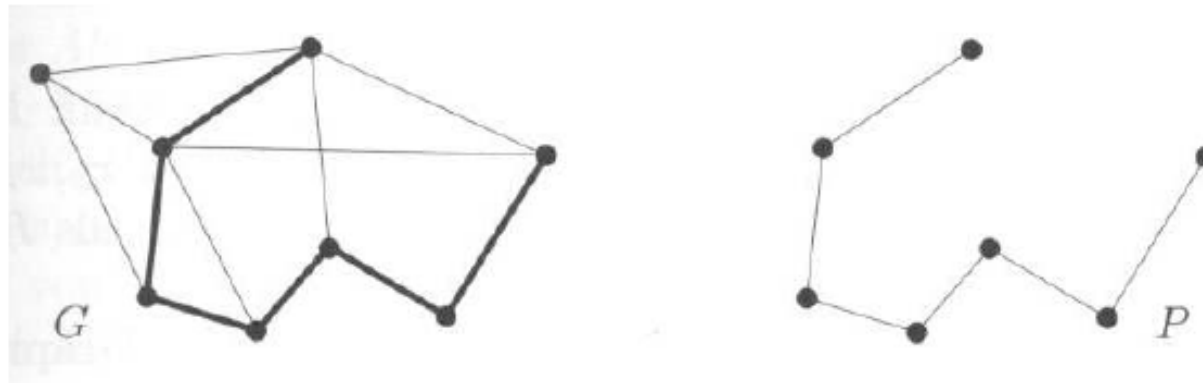


- Ein **Weg** ist ein nicht leerer Graph $G = (V, E)$ der Form

$$V = \{v_0, v_1, \dots, v_k\} \quad \text{und} \quad E = \{e_0, e_1, \dots, e_{k-1}, e_k\}$$

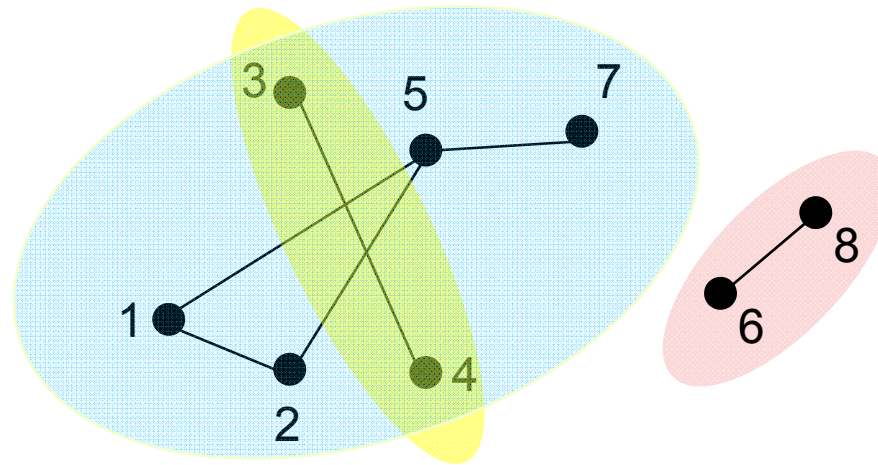
wobei die Knoten v_i paarweise verschieden sind. Die Knoten v_0 und v_k sind die Endknoten von G ; sind durch G verbunden.

- Die Anzahl der Kanten eines Weges ist seine Länge; der Weg der Länge k wird auch mit P^k bezeichnet.
- Ist $G = e_0, \dots, e_{k-1}$ ein Weg und $k \geq 3$, so ist der Graph $G' = G + e_{k-1}, e_0$ ein **Zyklus** (Kreis). Ein Graph ohne Zyklus heißt **azyklisch**.



Ein Weg $P = P^6$ in G

- Ein nicht leerer Graph heißt **zusammenhängend**, wenn er für je zwei seiner Knoten v_i, v_j einen $v_i - v_j$ -Weg enthält. Ein maximal zusammenhängender Teilgraph von G ist eine **(Zusammenhangs-)Komponente** von G .



- Gerichteter Weg** in G ist eine Folge (v_0, v_1, \dots, v_h) von Knoten in G , sodass (v_i, v_{i+1}) eine gerichtete Kante in G für $i = 0, \dots, h - 1$ ist.
- Gerichteter Weg heißt **Zyklus**, falls (v_h, v_0) eine gerichtete Kante in G ist.
- Gerichteter azyklischer Graph (DAG)** ist ein gerichteter Graph ohne Zyklus.

Allgemeine Eigenschaften:

1. Summe der Knotengrade $d_G(v)$ entspricht zweimal der Anzahl der Kanten $E(G)$

$$\sum_{v \in V} d_G(v) = 2 \cdot E(G)$$

2. In einem Graphen ist die Anzahl der Knoten mit ungeradem Grad gerade.

Beweis zu 2.:

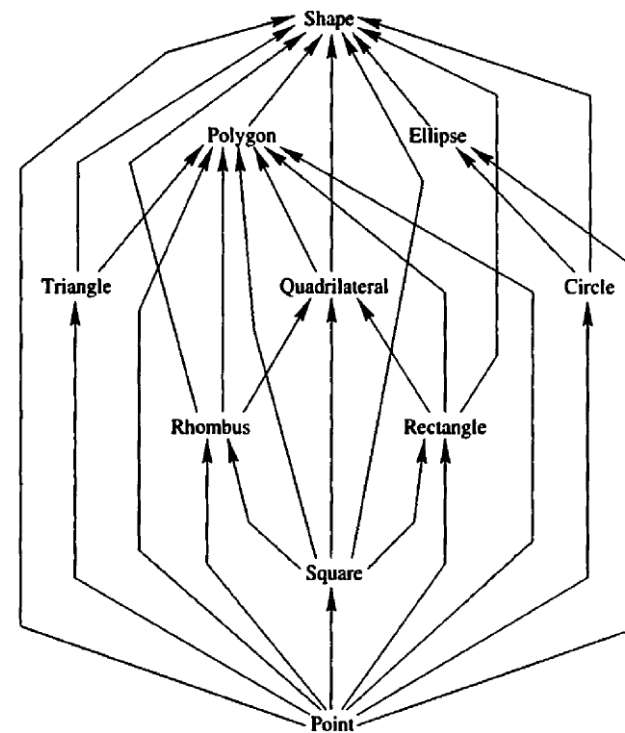
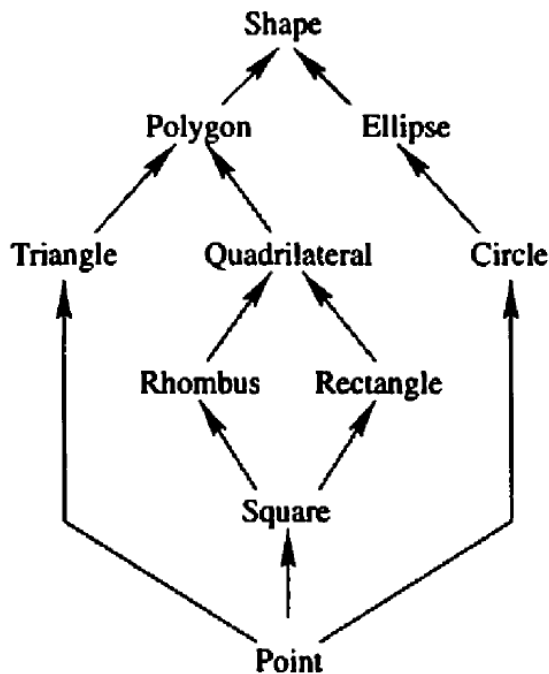
Aus 1. folgt, dass die Summe der Grade eine gerade Zahl ist. Zudem sei V_1 die Menge mit ungeradem und V_2 mit geradem Grad.

Dann gilt

$$2 \cdot E(G) = \sum_{v \in V} d_G(v) = \sum_{v \in V_1} d_G(v) + \sum_{v \in V_2} d_G(v)$$

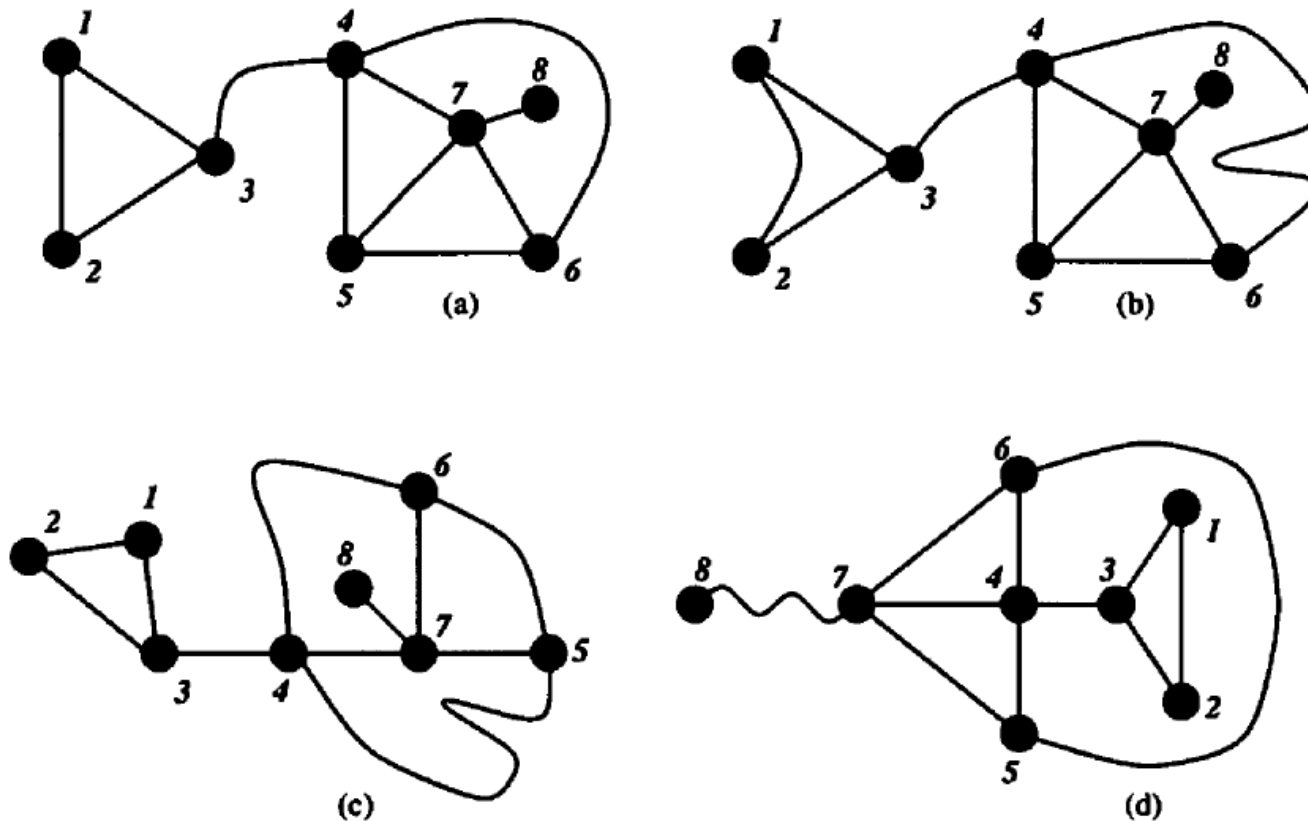
$$\underbrace{2 \cdot E(G) - \sum_{v \in V_2} d_G(v)}_{\text{gerade}} = \underbrace{\sum_{v \in V_1} d_G(v)}_{\substack{\text{gerade} \quad \text{ungerade}}}$$

- Eine Kante (v_i, v_j) eines Digraphen G heißt **transitiv**, falls es einen Weg von v_i nach v_j gibt, der nicht durch (v_i, v_j) läuft.
- **Transitiver Abschluss von G** ist ein Graph mit gleicher Knotenmenge und Kante (v_i, v_j) für jeden Weg von v_i nach v_j in G .



- Eine **Zeichnung** eines Graphen oder Digraphen G ist eine Abbildung G vom Graphen in die Ebene, die jedem Knoten v einen Punkt $G(v)$ und jeder Kante $\{v_i, v_j\}$ eine offene Jordankurve $G(v_i, v_j)$ zuordnet, welche die Punkte $G(v_i)$ und $G(v_j)$ als Endpunkte hat.
- Eine **Zeichnung** eines Graphen oder Digraphen heißt **planar**, wenn sich die Kurven zweier verschiedener Kanten höchstens in den Endpunkten schneiden.
- Ein **Graph** oder Digraph heißt **planar**, wenn es mindestens eine planare Zeichnung gibt.
- Eine planare Zeichnung eines Graphen oder Digraphen zerlegt die Ebene in topologisch verbundene Regionen, die **Facetten** genannt werden.
- Die unbegrenzte Facette wird **äußere Facette** genannt.

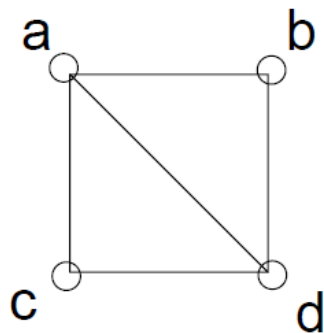
- Eine **planare Einbettung** eines Graphen oder Digraphen G ist eine Äquivalenzklasse von planaren Zeichnungen. Sie wird durch Festlegen der zirkulären Ordnungen der inzidenten Kanten an jedem Knoten festgelegt.
- **Beispiel:** a, b, c die gleiche, d eine andere Einbettung



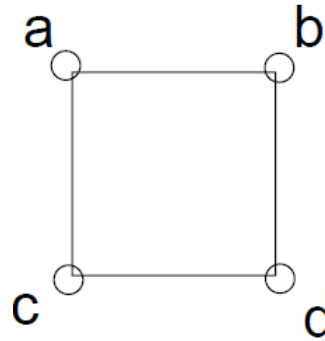
- Ein planarer Graph (gemäß Eulerformel) mit n Knoten hat höchstens $3n - 6$ Kanten

Euler-Formel: In einem planaren, zusammenhängenden Graphen mit n Knoten, m Kanten und f Flächen gilt $n - m + f = 2$.

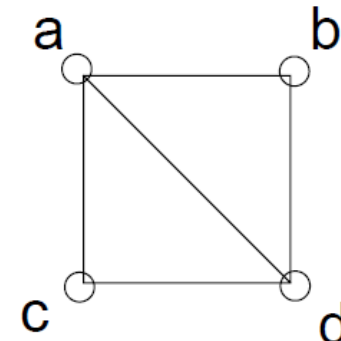
- **Euler-Weg/-Kreis** in G ist ein Weg, der jede Kante aus E genau einmal enthält.
 - G hat einen **Euler-Kreis** genau dann, wenn alle Knoten geraden Grad haben.
 - G hat einen **Euler-Weg**, der kein Kreis ist, genau dann, wenn G genau zwei Knoten mit ungeradem Grad hat.
- **Hamilton-Kreis** enthält jeden Knoten aus V genau einmal.



Euler-Weg
(a, b, d, a, c, d)



Euler-Kreis
(a, b, d, c, a)



Hamilton-Kreis
(a, b, d, c, a)

D.1. Graphentheorie

- D.1.1. Einleitung
- D.1.2. Gerichtete Graphen
- D.1.3. Ungerichtete Graphen
- D.1.4. Hypergraphen
- D.1.5. Speicherung
- D.1.6. Teilgraphen
- D.1.7. Pfade und Kreise
- D.1.8. Spezielle Eigenschaften

D.2. Zeichnen von Graphen

- D.2.1. Zeichenkonventionen
- D.2.2. Anforderungen
- D.2.3. Vorgehensweisen

D.3. Zeichenstrategien für Graphen

- D.3.1. Topologie-Form-Metrik-Ansatz
 - D.3.1.1. Verfeinerungsebenen
 - D.3.1.2. Algorithmus
 - D.3.1.3. Beispiel
- D.3.2. Hierarchischer Ansatz
 - D.3.2.1. Algorithmus
 - D.3.2.2. Beispiele
 - D.3.2.3. DAG
 - D.3.2.4. Bemerkungen
- D.3.3. Sichtbarkeitsansatz
 - D.3.3.1. Algorithmus
 - D.3.3.2. Beispiel
- D.3.4. Verfeinerungsansatz
 - D.3.4.1. Algorithmus
 - D.3.4.2. Beispiel

D.4. Schichtenweises Zeichnen allg. gerichteter Graphen

- D.4.1. Einleitung
- D.4.2. Algorithmus von Sugiyama
 - D.4.2.1. Übersicht
 - D.4.2.2. Entfernen von Zyklen (Schritt 1)
 - D.4.2.3. Schichtzuordnung (Schritt 2)
 - D.4.2.4. Kreuzungsreduktion (Schritt 3)
 - D.4.2.5. Waagrechte Koordinatenzuweisung (Schritt 4)
 - D.4.2.6. Bemerkungen

D.5. (Binäre) Bäume

- D.5.1. Konventionen
- D.5.2. Stufenweises Zeichnen
 - D.5.2.1. Einfacher Algorithmus
 - D.5.2.2. Algorithmus von Reingold/Tilford
- D.5.3. Horizontal-vertikal-Zeichnen
 - D.5.3.1. Algorithmus
 - D.5.3.2. Eigenschaften
- D.5.4. Zeichnen durch rekursives Drehen
 - D.5.4.1. Algorithmus
 - D.5.4.2. Eigenschaften
- D.5.5. Freie Bäume
- D.5.6. Bäume mit ausgedehnten Knoten

D.6. Vereinfachung von Polygonzügen

- D.6.1. Einführung
- D.6.2. min - # - Version (Imai, Iri)
- D.6.3. Fehlermaße
- D.6.4. min - e - Version
- D.6.5. Douglas-Peucker-Algorithmus

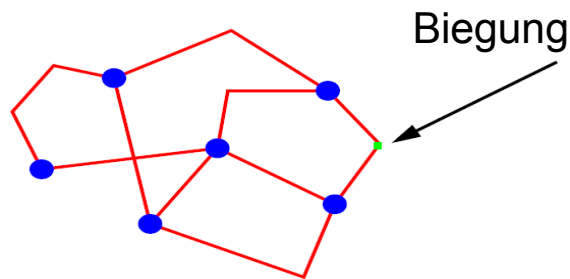
Gegenstand des Zeichnens von Graphen:

Verfahren, Algorithmen und Programme zur verständlichen graphischen Darstellung von Graphen

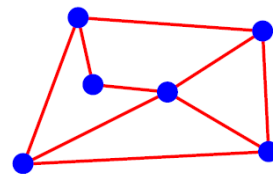
Zeichenkonventionen:

allgemeine Einschränkungen an die geometrische Repräsentation von Knoten und Kanten

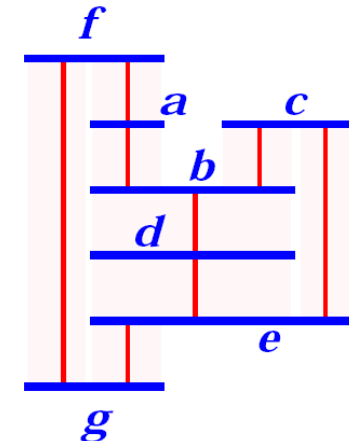
Polygonzugkanten



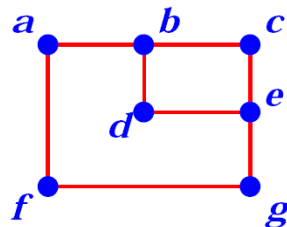
Streckenkanten



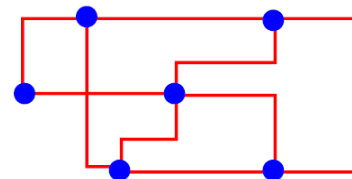
Starke Sichtbarkeitskanten



Orthogonale Streckenkanten



Orthogonale Kanten



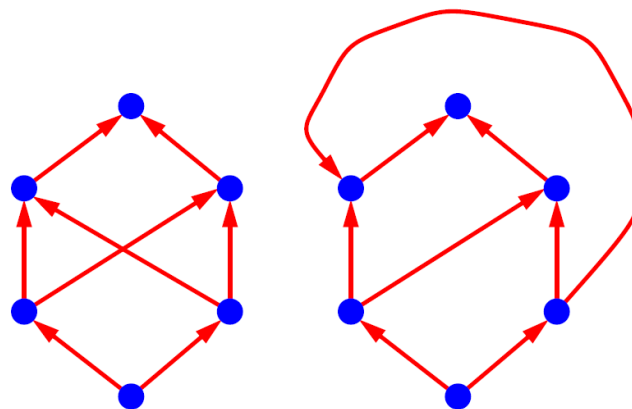
Zeichenkonventionen speziell für gerichtete Graphen

so zeichnen, dass alle Kanten in die gleiche Zeichenrichtung gerichtet sind, d.h. von links nach rechts oder von unten nach oben

Grund: dadurch werden hierarchische Beziehungen sichtbar

Schwierigkeit: die Konvention kann zu einem Konflikt mit dem Ziel der kantenschnittminimierenden Darstellung führen.

Beispiel:

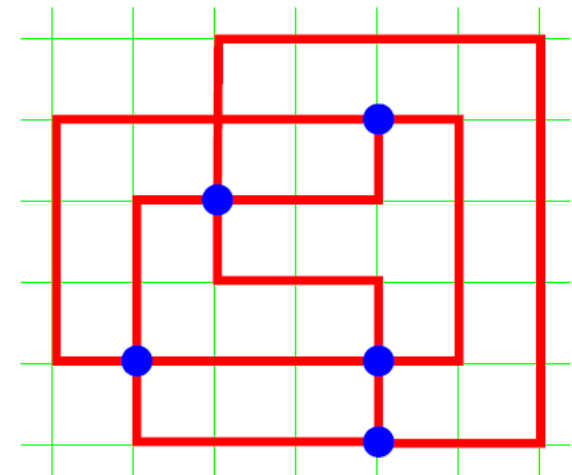
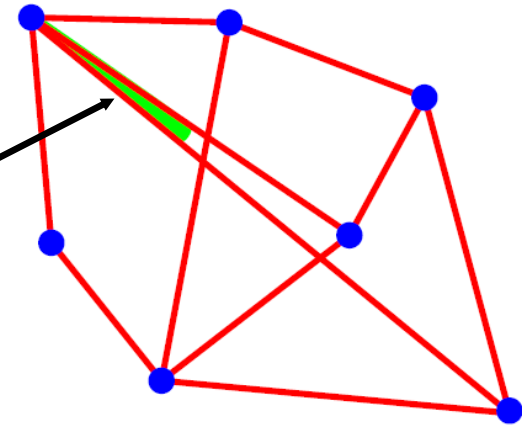


Auflösungsbezogene Anforderungen

- untere Schranke für den Knotenabstand
- untere Schranke für den Abstand zwischen Knoten und nicht inzidenten Kanten
- untere Schranke für den Winkel zwischen zwei aufeinanderfolgenden inzidenten Kanten

Grund: Beschränktheit der Auflösung des Zeichenmediums, z.B. eines Rasterbildschirms

Möglichkeit: Platzierung der Knoten auf einem Zeichengitter (gitterbasiertes Zeichnen)



Ästhetikbezogene Anforderungen zur besseren „Lesbarkeit“ des Graphen

- **Kantenschnitte.** Minimierung der Anzahl der Schnittpunkte von Kanten.
- **Fläche.** Graph wird auf möglichst kleiner Fläche gezeichnet. Die Fläche wird z.B. durch eine BoundingBox oder die konvexe Hülle gemessen.
- **Gesamtkantenlänge.** Gesamtkantenlänge wird minimiert. Dies ist nur bei skalierungsunabhängigen Zeichnungen sinnvoll.
- **Maximale Kantenlänge.** Länge der längsten Kante wird minimiert. Auch dies ist nur bei skalierungsunabhängigen Zeichnungen sinnvoll.
- **Gleichförmige Kantenlänge.** Varianz der Kantenlängen wird minimiert.
- **Gesamtanzahl Knicke.** Zahl aller Knicke aller Kanten soll minimiert werden. Dies ist besonders bei orthogonalem Zeichnen sinnvoll.
- **Maximale Knickanzahl.** Maximum von Knicken einer Kante wird minimiert.
- **Gleichmäßige Knickanzahl.** Varianz der Knickanzahl wird minimiert.
- **Winkelauflösung.** Kleinsten auftretenden Winkel zwischen zwei Kanten maximieren
- **Seitenverhältnis.** Seitenverhältnis des kleinsten umschriebenen Rechtecks soll nahe 1 sein.
- **Symmetrie.** Symmetrien des Graphen sollen möglichst gut abgebildet werden.

Anwendungsbezogene Anforderungen

Beeinflussung des Layouts des Graphen aufgrund der Kenntnis der Bedeutung einzelner Knoten, z.B. die nachfolgenden Konventionen

- **Zentrum.** Ein gegebener Knoten wird im Zentrum der Zeichnung fixiert.
- **Außen.** Ein bestimmter Knoten wird der äußeren Facette benachbart.
- **Cluster.** Eine gegebene Menge von Knoten wird nah beieinander angeordnet.
- **Vertikal-Horizontal-Folge.** Ein vorgegebener Weg wird vertikal/horizontal gezeichnet.
- **Form.** Zeichne einen gegebenen Teilgraphen mit vorgegebener Form

Bemerkung:

1. Die Anforderungen können möglicherweise nicht alle gleichzeitig optimiert werden.
2. Die Optimierung der Anforderungen ist teilweise algorithmisch aufwändig:
 - Minimierung der Kantenschnittanzahl ist NP-schwierig.
 - Minimierung der Anzahl der Kantenbiegungen in ebenen planaren Zeichnungen ist im Allgemeinen NP-schwierig.

aber:

- Das Testen eines Graphen auf Planarität (Zeichenbarkeit in der Ebene ohne Kantenschnitte) ist in linearer Zeit möglich.
- Für eine feste Einbettung kann die Anzahl der Kantenbiegungen in ebenen planaren Zeichnungen in polynomieller Zeit gefunden werden.

Aufgabe

Auffinden von Verfahren, die Graphzeichnungen generieren, die gegebene Konventionen erfüllen

Mögliche Vorgehensweisen

1. **algorithmisch:** Die ästhetischen Kriterien werden als Optimierungsprobleme formuliert, die algorithmisch exakt oder näherungsweise gelöst werden. Optimierungskriterien können die Minimierung der Anzahl der sich schneidenden Kanten oder der benötigten Zeichenfläche sein.

Vorteil: effizient

Nachteil: anwendungsspezifische Randbedingungen werden nicht in natürlicher Weise berücksichtigt.

Mögliche Vorgehensweisen

- 2. deklarativ:** Das Layout des Graphen wird durch anwendungsspezifische Nebenbedingungen und Zielfunktionen beschrieben. Das Layout (d.h. die Zeichnung) wird durch Lösen des Randbedingungssystems (engl. constraint system) erzeugt.

Vorteil: hohe Ausdruckskraft

Nachteile:

- Die Beschreibung einiger natürlicher ästhetischer Kriterien erfordert komplizierte Randbedingungen.
- Allgemeine Löser für Nebenbedingungssysteme sind rechnerisch aufwändig

Literatur:

- G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing, Prentice Hall, 1999
- Graph Drawing Server: loki.cs.brown.edu:8081/graphserver/ oder [rt/cs.brown.edu](http://rt.cs.brown.edu)

D.1. Graphentheorie

- D.1.1. Einleitung
- D.1.2. Gerichtete Graphen
- D.1.3. Ungerichtete Graphen
- D.1.4. Hypergraphen
- D.1.5. Speicherung
- D.1.6. Teilgraphen
- D.1.7. Pfade und Kreise
- D.1.8. Spezielle Eigenschaften

D.2. Zeichnen von Graphen

- D.2.1. Zeichenkonventionen
- D.2.2. Anforderungen
- D.2.3. Vorgehensweisen

D.3. Zeichenstrategien für Graphen

- D.3.1. Topologie-Form-Metrik-Ansatz
 - D.3.1.1. Verfeinerungsebenen
 - D.3.1.2. Algorithmus
 - D.3.1.3. Beispiel
- D.3.2. Hierarchischer Ansatz
 - D.3.2.1. Algorithmus
 - D.3.2.2. Beispiele
 - D.3.2.3. DAG
 - D.3.2.4. Bemerkungen
- D.3.3. Sichtbarkeitsansatz
 - D.3.3.1. Algorithmus
 - D.3.3.2. Beispiel
- D.3.4. Verfeinerungsansatz
 - D.3.4.1. Algorithmus
 - D.3.4.2. Beispiel

D.4. Schichtenweises Zeichnen allg. gerichteter Graphen

- D.4.1. Einleitung
- D.4.2. Algorithmus von Sugiyama
 - D.4.2.1. Übersicht
 - D.4.2.2. Entfernen von Zyklen (Schritt 1)
 - D.4.2.3. Schichtzuordnung (Schritt 2)
 - D.4.2.4. Kreuzungsreduktion (Schritt 3)
 - D.4.2.5. Waagrechte Koordinatenzuweisung (Schritt 4)
 - D.4.2.6. Bemerkungen

D.5. (Binäre) Bäume

- D.5.1. Konventionen
- D.5.2. Stufenweises Zeichnen
 - D.5.2.1. Einfacher Algorithmus
 - D.5.2.2. Algorithmus von Reingold/Tilford
- D.5.3. Horizontal-vertikal-Zeichnen
 - D.5.3.1. Algorithmus
 - D.5.3.2. Eigenschaften
- D.5.4. Zeichnen durch rekursives Drehen
 - D.5.4.1. Algorithmus
 - D.5.4.2. Eigenschaften
- D.5.5. Freie Bäume
- D.5.6. Bäume mit ausgedehnten Knoten

D.6. Vereinfachung von Polygonzügen

- D.6.1. Einführung
- D.6.2. min - # - Version (Imai, Iri)
- D.6.3. Fehlermaße
- D.6.4. min - e - Version
- D.6.5. Douglas-Peucker-Algorithmus

Grundlegenden Verfahren / Strategien zum Zeichnen von Graphen:

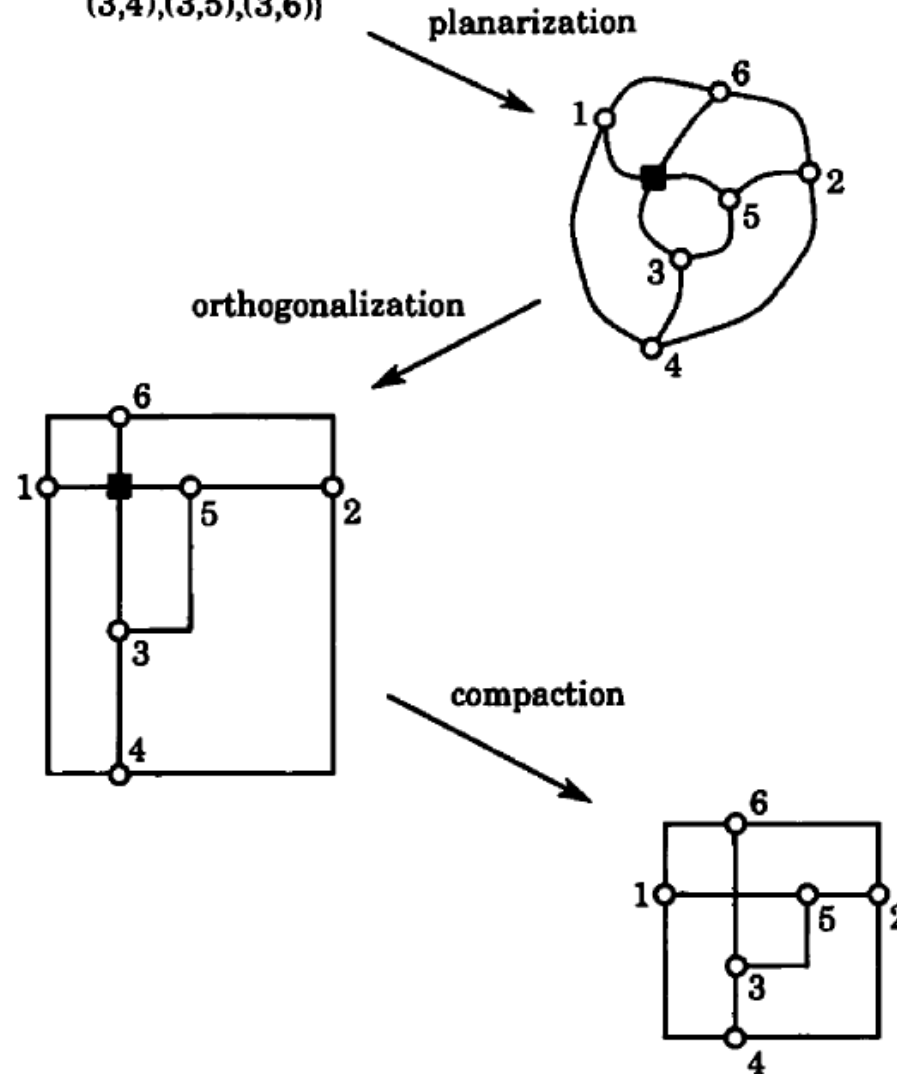
1. Topologie-Form-Metrik-Ansatz
2. Hierarchischer Ansatz
3. Sichtbarkeitsansatz
4. Verfeinerungsansatz

Algorithmus im Hinblick auf eine orthogonale Gitterzeichnung und basierend auf drei Verfeinerungsebenen:

1. **Topologie:** Zwei orthogonale Zeichnungen haben gleiche Topologie, wenn sie durch stetige Deformation ohne Vertauschen von Facetten ineinander überführt werden können.
2. **Form:** Zwei orthogonale Zeichnungen haben gleiche Form, wenn sie
 - gleiche Topologie haben und
 - eine Zeichnung aus der anderen nur durch Längenänderungen der Liniensegmente hervorgeht.
3. **Metrik:** Zwei orthogonale Zeichnungen haben gleiche Metrik, wenn sie
 - kongruent sind, also durch
 - Verschieben und Drehen ineinander überführt werden können.

Algorithmische Umsetzung entsprechend der drei Verfeinerungsebenen:

1. **Planarisierung** legt die Einbettung des Graphen, also seine Topologie fest und beruht auf der Minimierung der Anzahl der Kantenschnitte:
 - Maximalen planaren Untergraph ermitteln
 - Verbleibende Kanten einzeln eingefügt, mit möglichst wenig Überschneidungen
 - Dummy-Knoten an verbleibenden Überschneidungen einfügen.
2. **Orthogonalisierung** legt die Form fest:
 - Orthogonale Repräsentation des Graphen
 - Knoten ohne Koordinaten
 - Kanten lediglich Winkellisten, zur Festlegung ihrer Knicke
 - Minimierung der Knicke ist in der Regel das Hauptziel
3. **Kompaktifizierung** legt endgültige Koordinaten der Kanten und der Kantenknicke fest: In der Regel wird die Fläche des Graphen minimiert.

Beispiel: $V = \{1, 2, 3, 4, 5, 6\}$
 $E = \{(1, 4), (1, 5), (1, 6),$
 $(2, 4), (2, 5), (2, 6),$
 $(3, 4), (3, 5), (3, 6)\}$ 

Ansatz speziell für das Layout von DAGs.

1. Schichtungsschritt

- verwandelt einen DAG in einen geschichteten Digraphen, bei dem jedem Knoten eine Schicht L_i zugeordnet wird, sodass für alle Kanten (v_l, v_k) gilt: $v_l \in L_i \wedge v_k \in L_j$ mit $i < j$.
- Nach Einfügen von Dummy-Knoten auf Kanten, die über mehr als eine Ebene laufen - sodass jede Kante von einer Schicht i zur Schicht $i + 1$ läuft - ist dies ein echt geschichteter Digraph.

2. Bei Schnittreduzierung erhält jede Schicht eine Reihenfolge (Ordnung)

- Reihenfolge bestimmt Topologie des Layouts.
- Dabei wird Anzahl der Kantenschnitte minimiert.

3. Festlegen der x-Koordinaten legt x-Koordinaten der Knoten in den Ebenen fest

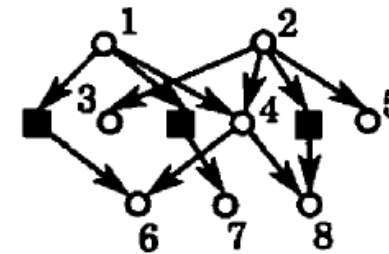
- Für endgültige Zeichnung werden Dummy-Knoten durch Knicke ersetzt
- Knicke können minimiert oder Symmetrien betont werden

Beispiel 1:

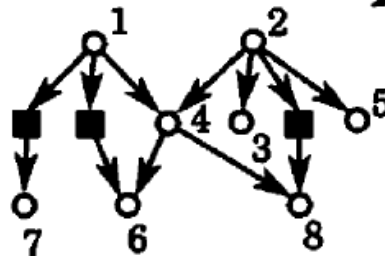
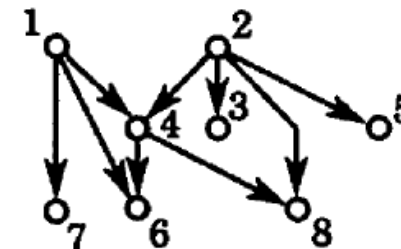
$$V = \{1,2,3,4,5,6,7,8\}$$

$$E = \{(1,4), (1,6), (1,7), \\ (2,3), (2,4), (2,5), \\ (2,8), (4,6), (4,8)\}$$

layer assignment



crossing reduction

 x -coordinate assignment

Problem: Generelle Digraphen enthalten Zyklen

Entfernen von Zyklen

Invertiere die Richtung einiger Kanten, um den Graph azyklisch zu machen.

Problem: Minimale Anzahl von Rückwärtskanten

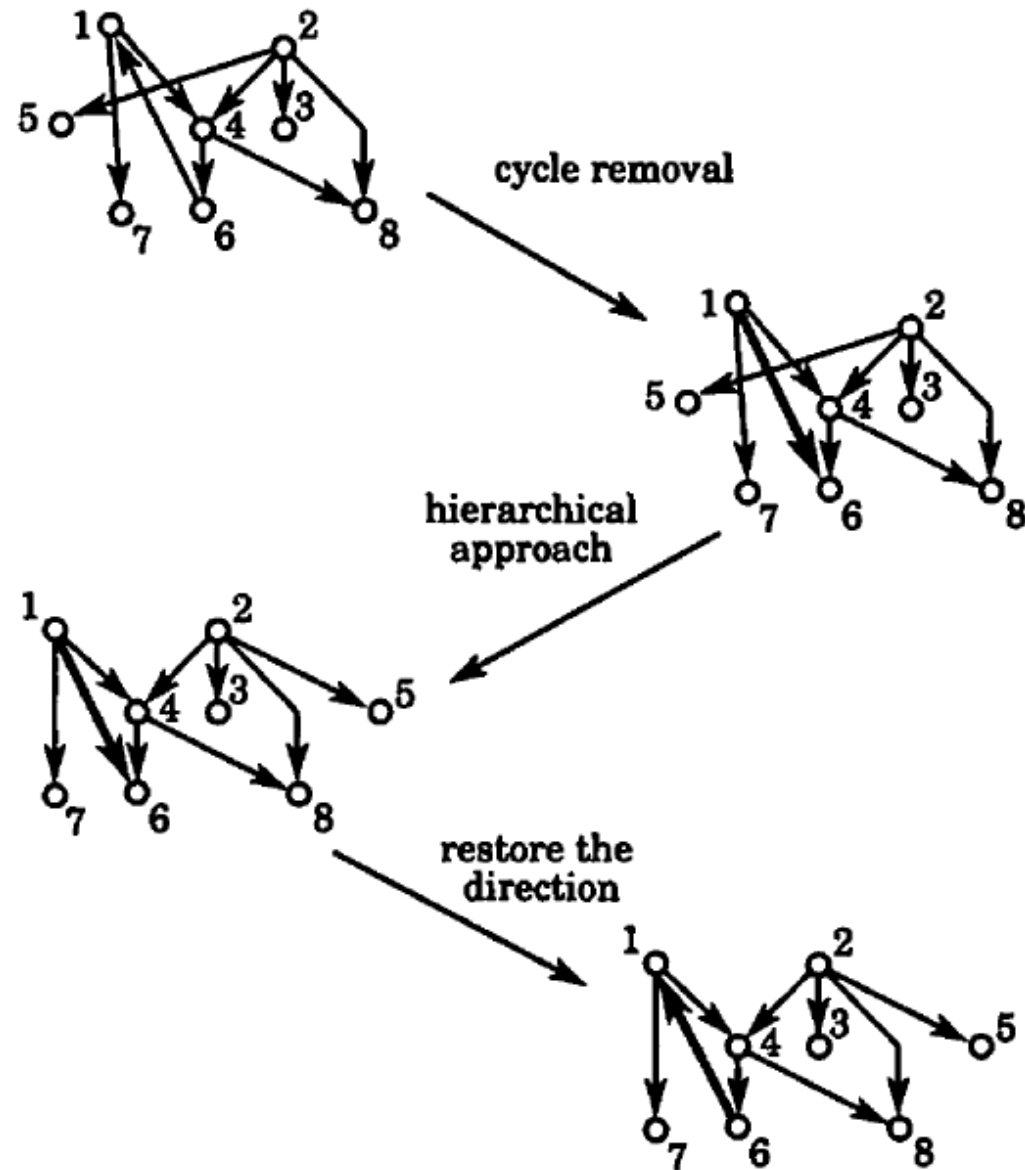
Gegeben: Ein gerichteter Graph $G = (V, E)$.

Gesucht: Eine Teilmenge R von E kleinster Mächtigkeit, so dass G azyklisch wird, wenn die Kanten von R invertiert werden.

Schwierigkeit: Das Problem ist NP-schwierig.

Näherungsweise Lösung: Greedy-Algorithmus zur Zyklentfernung
(s. Kapitel D.4.2. 2 zum Algorithmus von Sugiyama)

Beispiel 2:
Genereller Digraph mit Zyklen



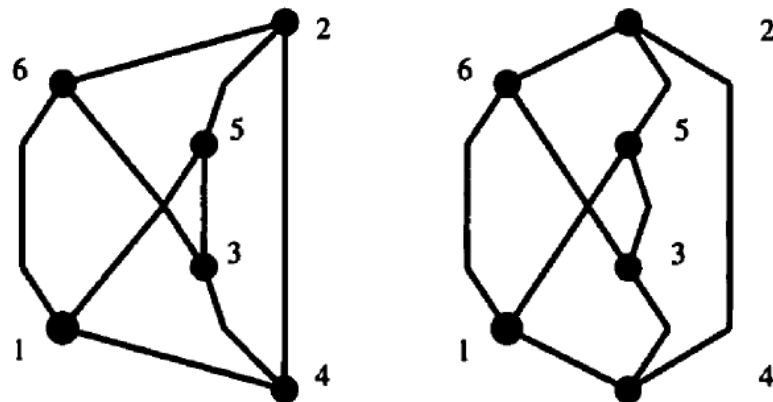
Bemerkungen:

- Problem der **Schnittminimierung** zwischen zwei Ebenen ist bereits **NP-hart**: Einsatz von Heuristiken (s. Kapitel D.4.2. Algorithmus von Sugiyama)
K. Sugiyama, S. Tagawa, and M. Toda, „Methodes for Visual Understanding of Hierarchical Systems Structures“, IEEE Trans. Systems, Man, and Cybernetics, vol. 11, no. 2, pp. 109-125, 1989
- Heuristik legt eine **Ordnung der ersten und letzten Ebene fest**: jeder Knoten soll im Schwerpunkt seiner Nachbarn (im Graphen) liegen: lineares Gleichungssystem
W. Tutte, „How to Draw a Graph“, Proc. London Math. Soc., vol. 3, no. 13, pp. 743-768, 1963
- Vergleich von Heuristiken
M. Laguna and R. Marti, „Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization“, URL: <http://www-bus.colorado.edu/Faculty/Laguna/>, 1999.

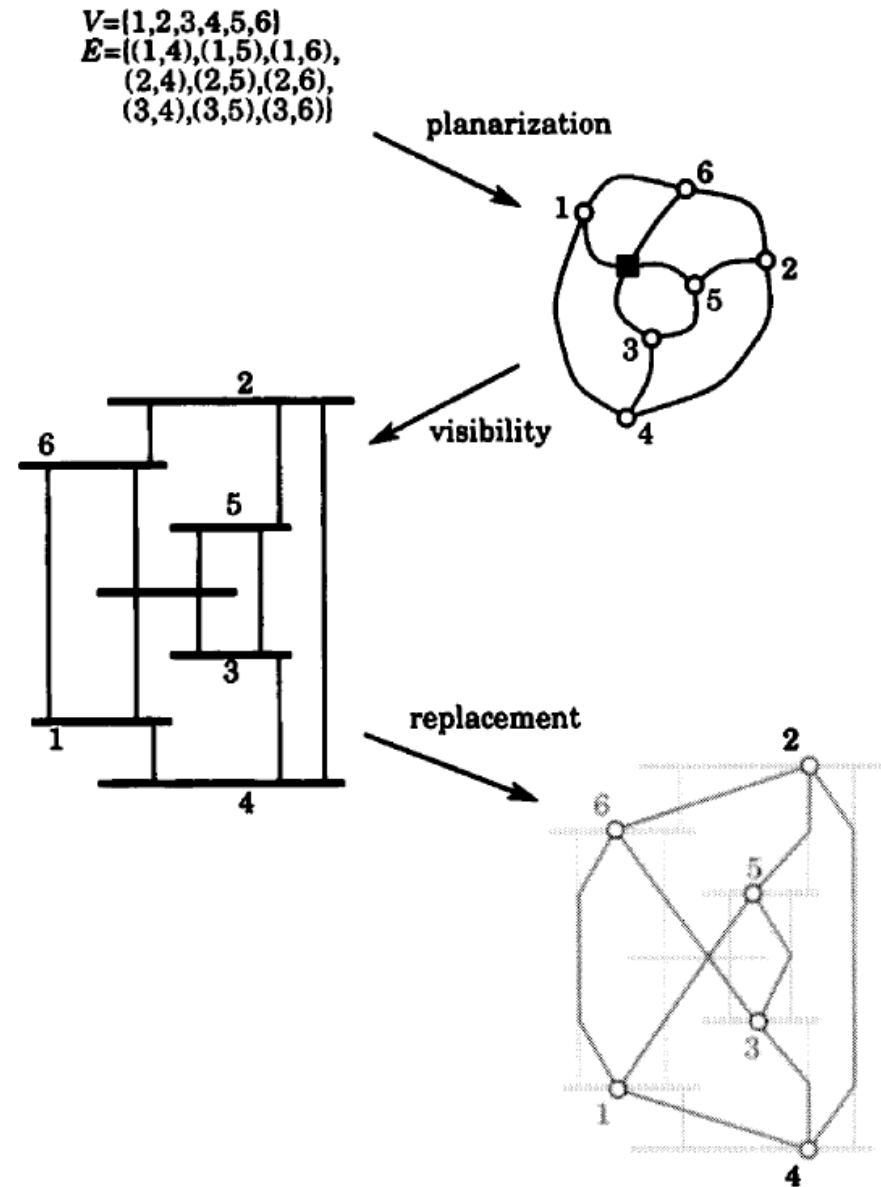
Ansatz dient dem Anordnen beliebiger Graphen mit Polylinien.

1. **Planarisierung** gleicht dem Topologie-Form-Metrik-Ansatz;
Ziel ist Schnittvermeidung
2. **Sichtbarkeitsschritt** schafft eine Sichtbarkeitsrepräsentation des Graphen
 - Jedem **Knoten** wird **horizontales** und jeder **Kante** (v_i, v_j) **vertikales Segment** zugeordnet
 - Kantensegment hat Anfang und Ende innerhalb der Knotensegmente
 - Kantensegmente schneiden einander nicht
3. **Ersetzungsschritt** erzeugt endgültige Zeichnung des Graphen
 - Ersetzen jedes **horizontalen** Segments durch einen **Punkt**
 - Ersetzen jedes **vertikalen** Segments durch eine **Polylinie**
 - Mögliche Strategien / Ästhetikkriterien: Knickminimierung, Symmetriebetonung, gleichmäßige Verteilung der Knoten

Beispiel:



Alternative Ersetzungsstrategien

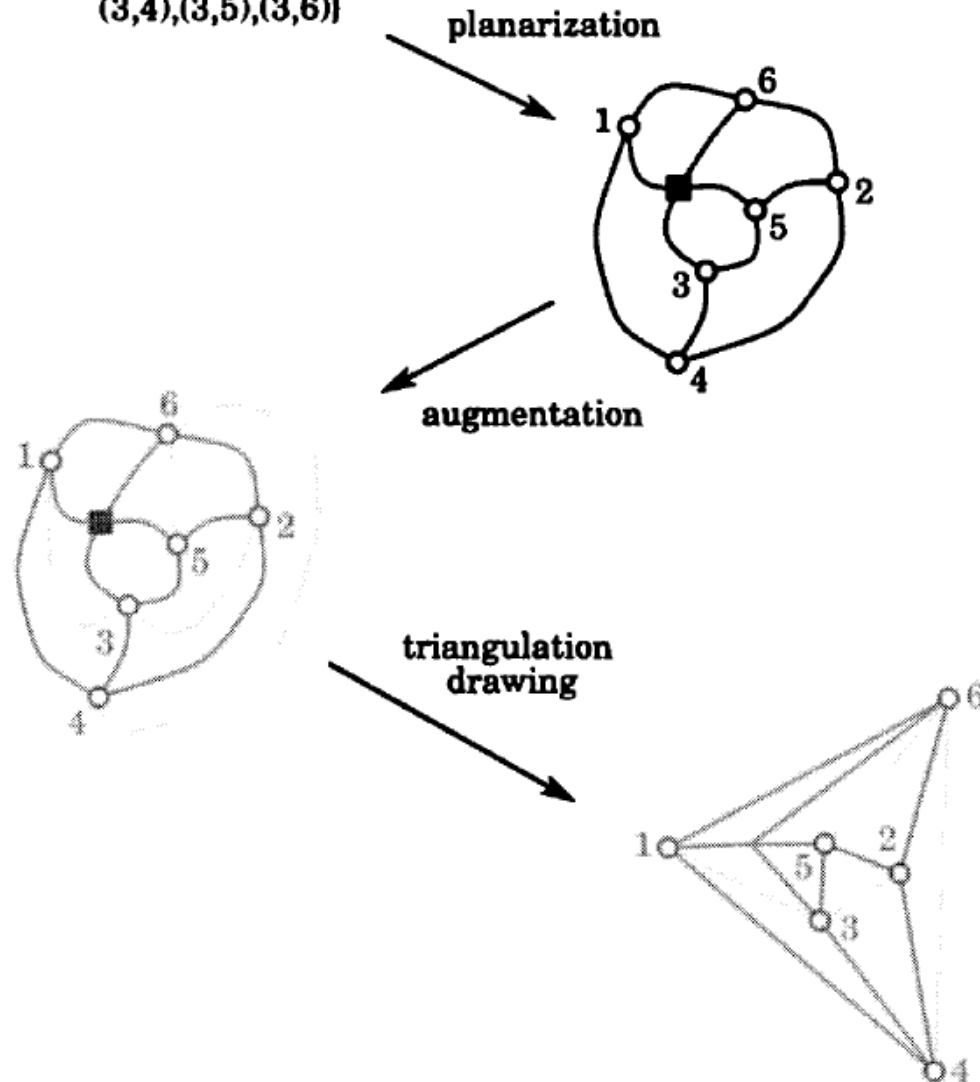


Betrachtet Polylinienzeichnungen

1. **Planarisierung** gleicht Topologie-Form-Metrik-Ansatz; Schnittvermeidung
2. **Verfeinerung** fügt geeignete Anzahl von Kanten (evtl. auch Knoten) in Einbettung ein, um einen maximalen planaren Graphen zu erzeugen:
 - Alle Facetten haben dann drei Kanten - Triangulierung
 - Qualität des Zeichnens maximaler Graphen hängt in der Regel vom Knotengrad ab
 - Minimierung des maximalen Knotengrads
3. **Triangulierungszeichnung** erzeugt Zeichnung durch
 - „Auslegen“ aller Dreiecke und
 - anschließendes Entfernen der Dummy-Kanten und ggf. Dummy-Knoten

Bemerkung:

Alle Algorithmen nutzen spezielle Triangulierungseigenschaften durch überschneidende aufspannende Bäume oder kanonische Knotenaufzählungen (Beim Entfernen der Dummy-Knoten können Knicke und Schnitte entstehen.)

Beispiel: $V = \{1, 2, 3, 4, 5, 6\}$
 $E = \{(1, 4), (1, 5), (1, 6),$
 $(2, 4), (2, 5), (2, 6),$
 $(3, 4), (3, 5), (3, 6)\}$ 

D.1. Graphentheorie

- D.1.1. Einleitung
- D.1.2. Gerichtete Graphen
- D.1.3. Ungerichtete Graphen
- D.1.4. Hypergraphen
- D.1.5. Speicherung
- D.1.6. Teilgraphen
- D.1.7. Pfade und Kreise
- D.1.8. Spezielle Eigenschaften

D.2. Zeichnen von Graphen

- D.2.1. Zeichenkonventionen
- D.2.2. Anforderungen
- D.2.3. Vorgehensweisen

D.3. Zeichenstrategien für Graphen

- D.3.1. Topologie-Form-Metrik-Ansatz
 - D.3.1.1. Verfeinerungsebenen
 - D.3.1.2. Algorithmus
 - D.3.1.3. Beispiel
- D.3.2. Hierarchischer Ansatz
 - D.3.2.1. Algorithmus
 - D.3.2.2. Beispiele
 - D.3.2.3. DAG
 - D.3.2.4. Bemerkungen
- D.3.3. Sichtbarkeitsansatz
 - D.3.3.1. Algorithmus
 - D.3.3.2. Beispiel
- D.3.4. Verfeinerungsansatz
 - D.3.4.1. Algorithmus
 - D.3.4.2. Beispiel

D.4. Schichtenweises Zeichnen allg. gerichteter Graphen

- D.4.1. Einleitung
- D.4.2. Algorithmus von Sugiyama
 - D.4.2.1. Übersicht
 - D.4.2.2. Entfernen von Zyklen (Schritt 1)
 - D.4.2.3. Schichtzuordnung (Schritt 2)
 - D.4.2.4. Kreuzungsreduktion (Schritt 3)
 - D.4.2.5. Waagrechte Koordinatenzuweisung (Schritt 4)
 - D.4.2.6. Bemerkungen

D.5. (Binäre) Bäume

- D.5.1. Konventionen
- D.5.2. Stufenweises Zeichnen
 - D.5.2.1. Einfacher Algorithmus
 - D.5.2.2. Algorithmus von Reingold/Tilford
- D.5.3. Horizontal-vertikal-Zeichnen
 - D.5.3.1. Algorithmus
 - D.5.3.2. Eigenschaften
- D.5.4. Zeichnen durch rekursives Drehen
 - D.5.4.1. Algorithmus
 - D.5.4.2. Eigenschaften
- D.5.5. Freie Bäume
- D.5.6. Bäume mit ausgedehnten Knoten

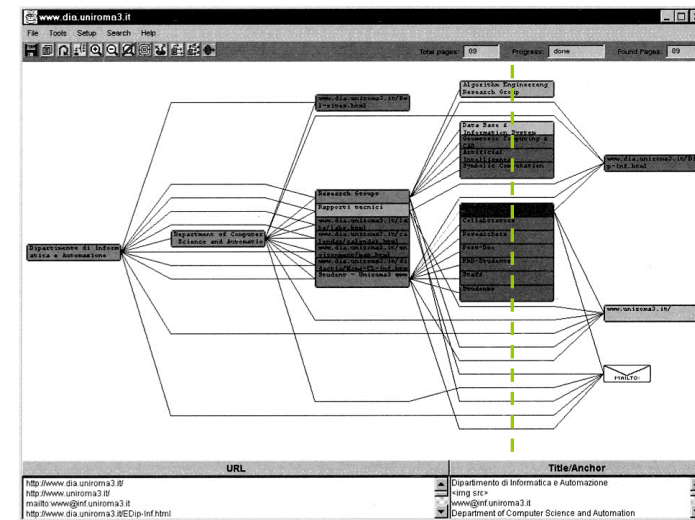
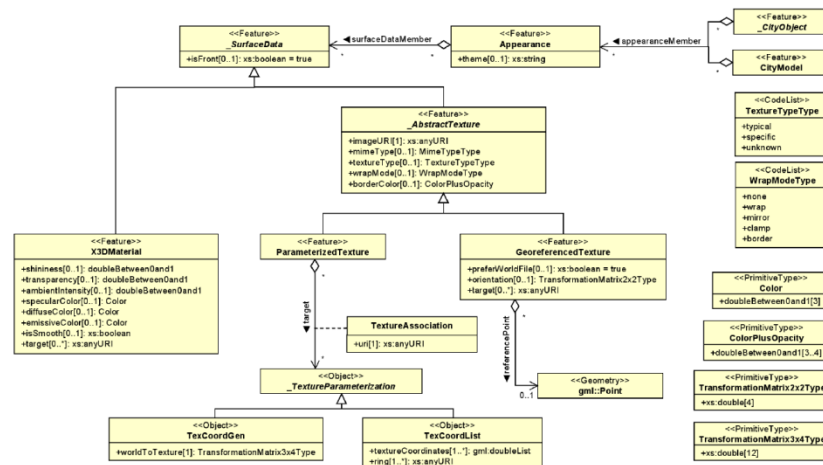
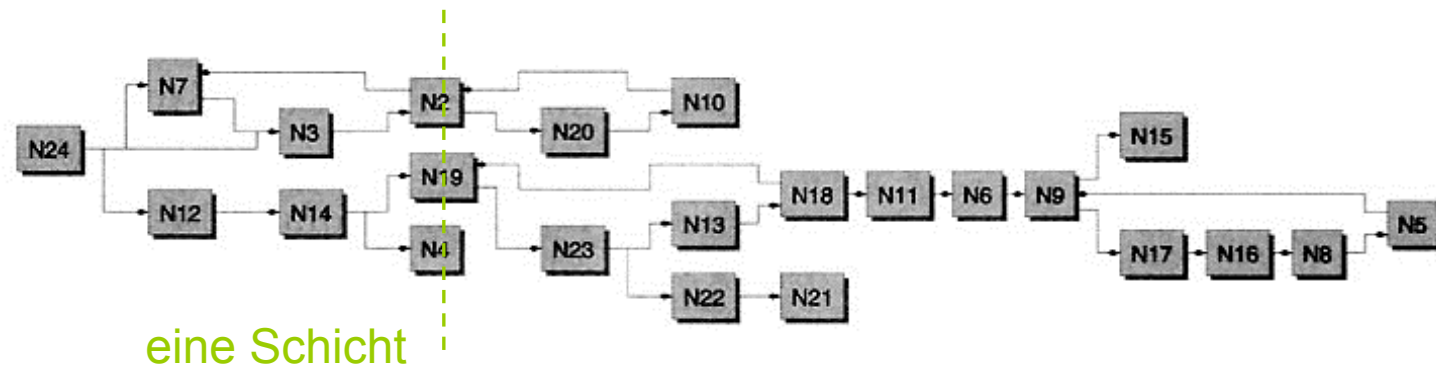
D.6. Vereinfachung von Polygonzügen

- D.6.1. Einführung
- D.6.2. min - # - Version (Imai, Iri)
- D.6.3. Fehlermaße
- D.6.4. min - e - Version
- D.6.5. Douglas-Peucker-Algorithmus

Schichtenweises Zeichnen von allgemeinen gerichteten Graphen

- Knoten werden auf Schichten angeordnet
- hohe Anwendungsbedeutung: Bioinformatik, Betriebswirtschaft, Politik, Medizin, Projekt-Management, (Geo)Informatik, usw.

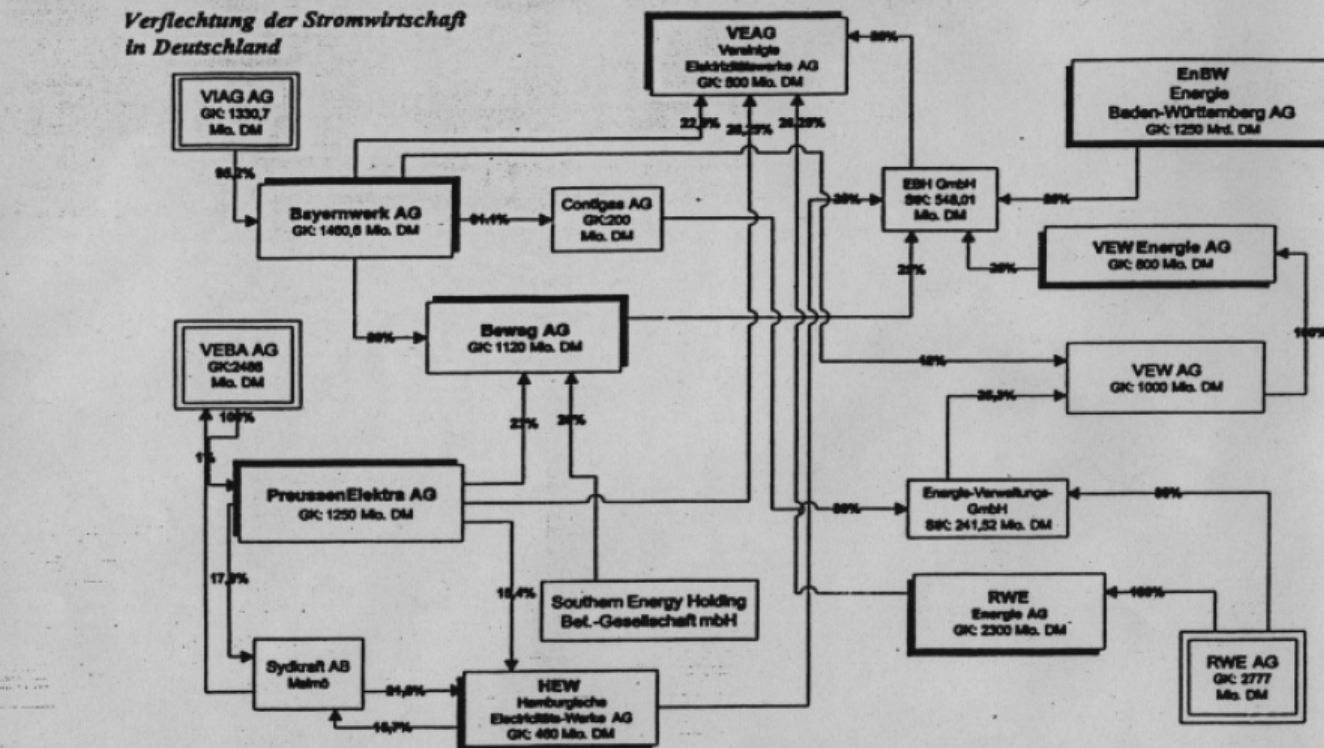
Beispiele:



■ **Mysterien und Leberwurstbrote oder: Die wirrsten Grafiken der Welt (8)****Die Verflechtung der Stromwirtschaft (nach Michael Stelte)** taz

Wie stark verflochten die Stromwirtschaft in Deutschland ist, zeigt eine bildschöne Grafik aus Michael Steltes Berliner Energiedatenbank: Die Hamburgische Electricitäts-Werke AG ist beispielsweise mit 15,7 Prozent an der Sydkraft AB Malmö beteiligt, diese mit einem Prozent an der Veba AG, diese mit 100 Prozent an der Veag AG, diese mit 100 Prozent an der PreussenElektra AG, diese mit 17,6 Prozent an der Sydkraft AB Malmö und diese wiederum mit 21,8 Prozent an der Hamburgischen Electricitäts-Werke AG. Ob es wohl Steuerprüfer gibt, die es wagen, sich solche komplizierten Besitzverhältnisse einzuprägen und mit Taschenrechner, Aspirin und Leberwurstbrot bewaffnet in die Mysterien der Stromwirtschafts-Buchhaltung einzudringen? Oder überlässt das Finanzamt diese Arbeit lieber Sisyphus? Um sich stattdessen wehrlosen Kleinsparern zuzuwenden?

Das wäre verständlich, denn die Verflechtung der Stromwirtschaft ist nahezu unentwirrbar, auch wenn die Grafik auf den ersten Blick sehr übersichtlich wirkt. Einmal angenommen, die Veag erwirtschaftet zehn Mark Gewinn. Dann gehören 25 Prozent davon der EBH GmbH (= 2,50 DM), davon 25 Prozent der VEW Energie AG (= 0,625 DM), davon 100 Prozent der VEW AG (= 0,625 DM) und davon 12 Prozent der Bayern-



werk AG (= 0,075 DM). Aber 25,3 Prozent von den 0,625 DM gehören der Energie-Verwaltungs-GmbH (= 0,158125 DM), davon 30 Prozent der Contigas AG (= 0,0474375 DM) und davon 91,1 Prozent abermals der Bayernwerk AG (= 0,0432155625 DM).

Damit hat die Bayernwerk AG ihren Gewinn also bereits auf satte 0,1182155625 DM aufgestockt. Aber die Bayernwerk ist ja zudem auch noch mit weiteren 22,5 Prozent direkt an der Veag beteiligt, so dass von den zehn Mark noch einmal 2,25 DM abfallen und sich so-

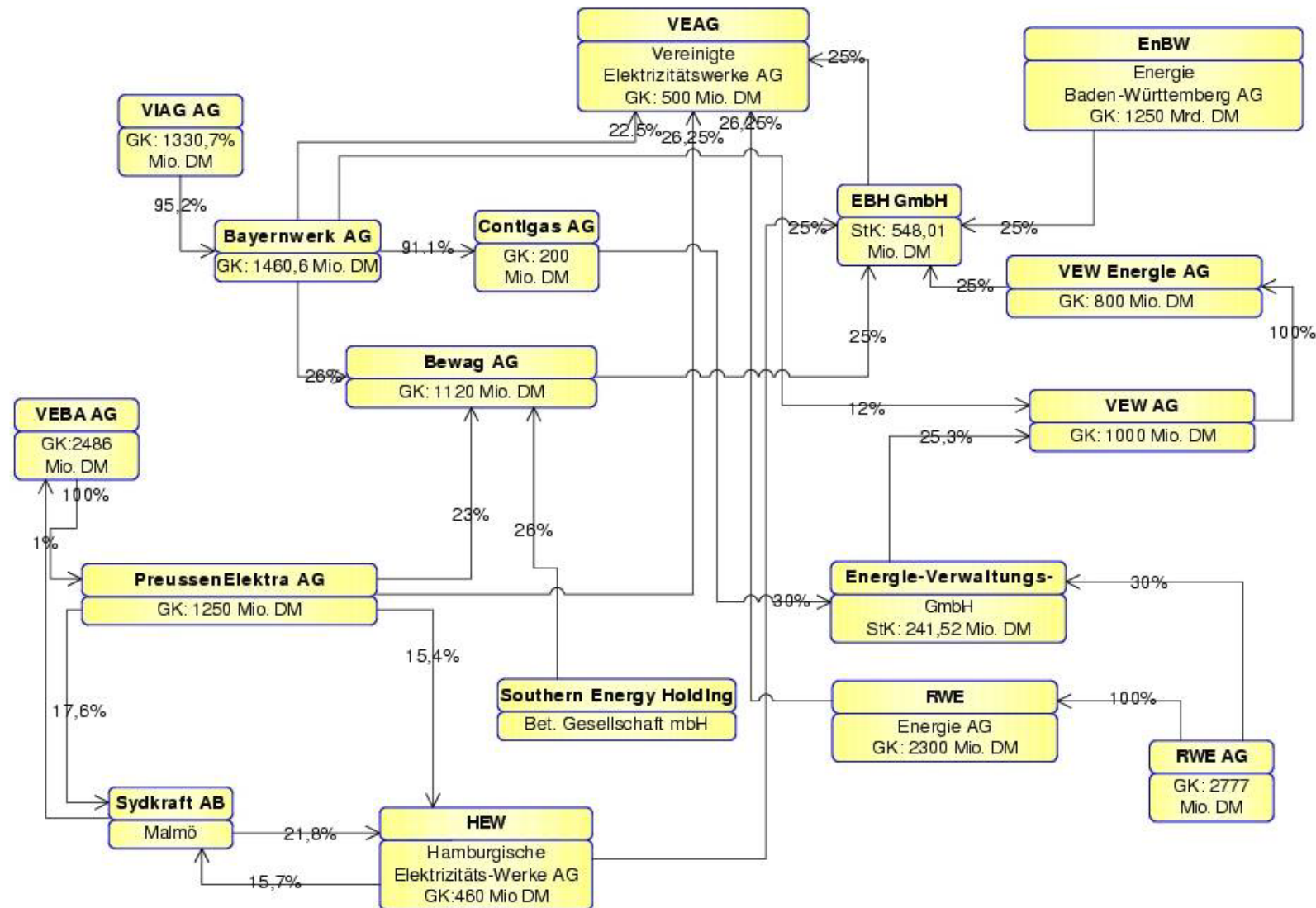
mit eine Gewinnsumme von 2,3682155625 DM ergibt.

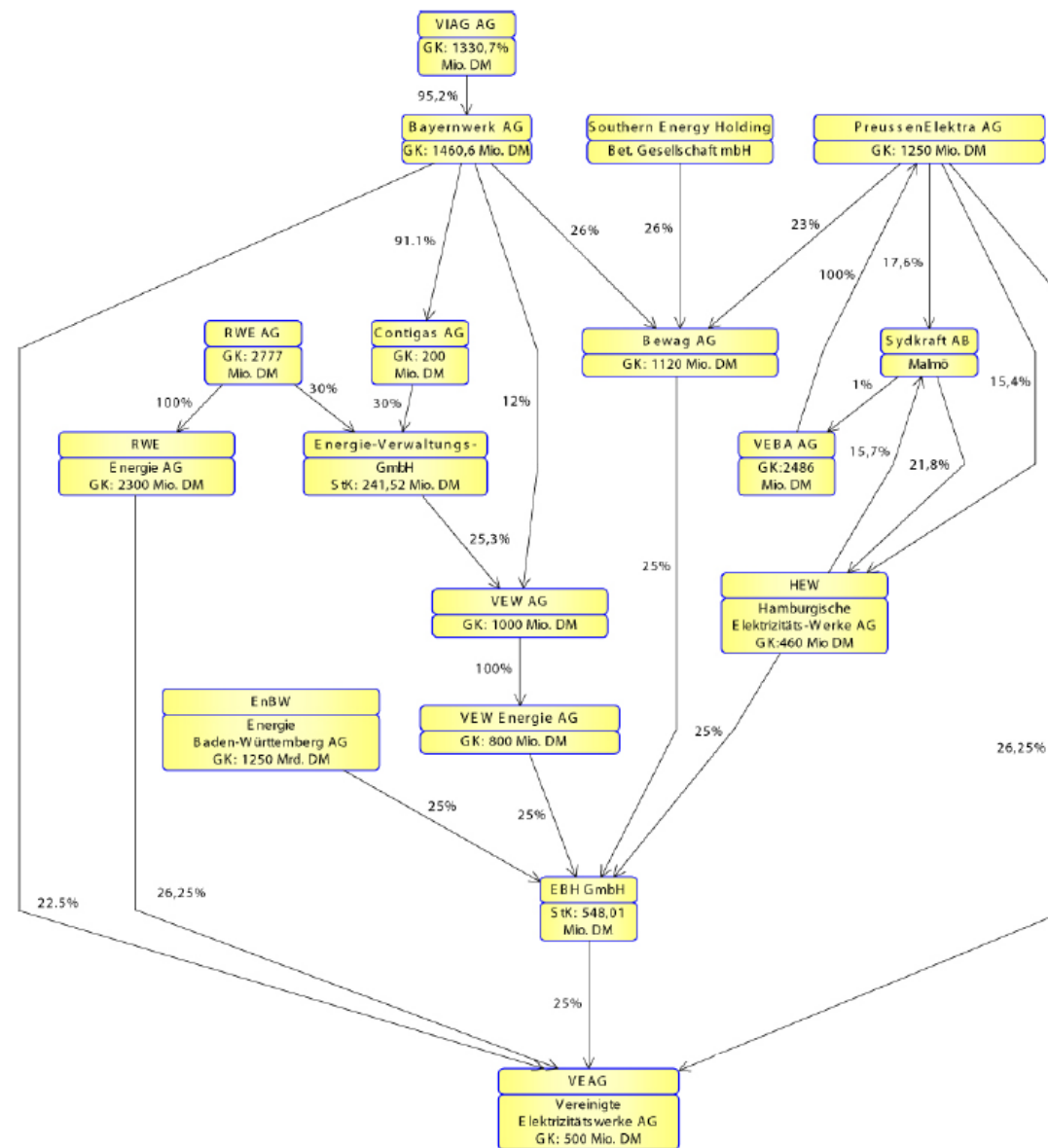
Aufgaben:

1) Reche aus, wieviel noch dazu kommt, da die Bayernwerk AG ja auch über ihre Beteiligung an der Bewag an der EBH GmbH beteiligt ist!

2) Wieviel schuldet die Bayernwerk AG letztlich der Viag AG?
3) Ruf bei der Bayernwerk AG in München an (Durchwahl Hauptverwaltung [089] 12541) und erkundige dich danach, ob deine Rechnung stimmt!

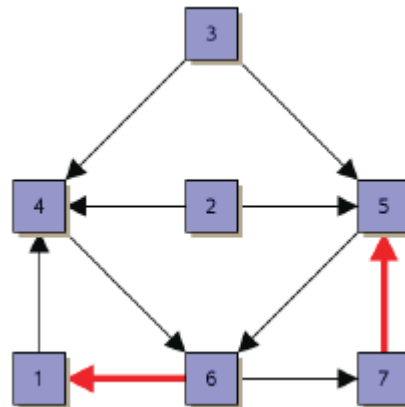
Gerhard Henschel



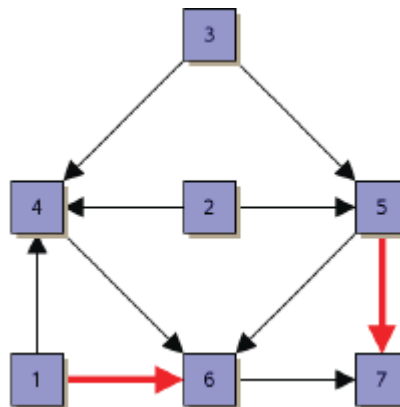


Hierarchischer Algorithmus von Sugiyama, Tagawa, Toda

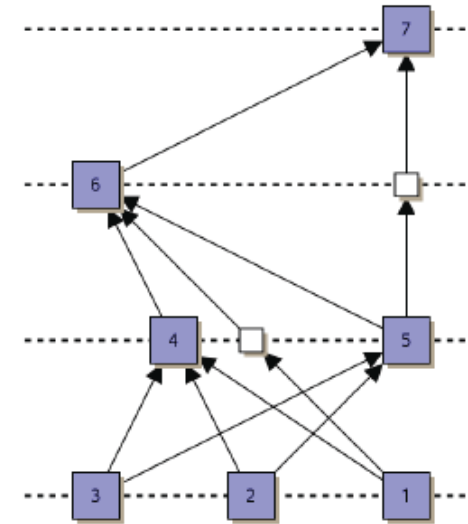
Schichtenweises Zeichnen von gerichteten Graphen



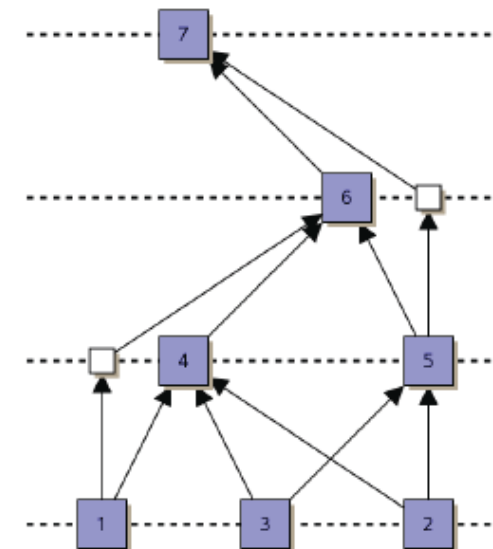
1. **Entfernen von Zyklen:** Invertiere die Richtung einiger Kanten, um den Graph azyklisch zu machen.



- Schichtzuordnung:** Weise die Knoten waagrechten Schichten zu (y-Koordinate des Knotens).

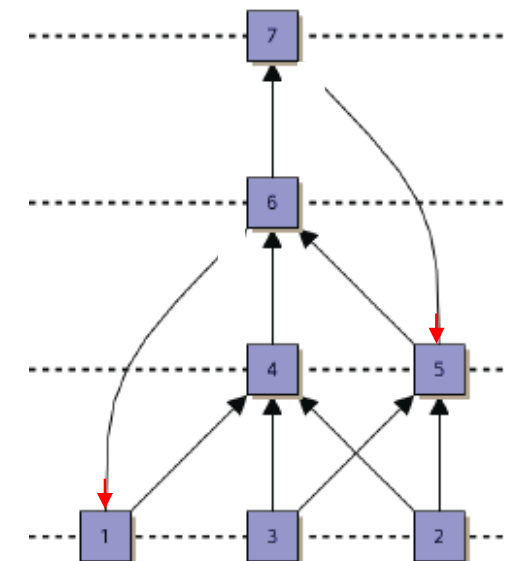
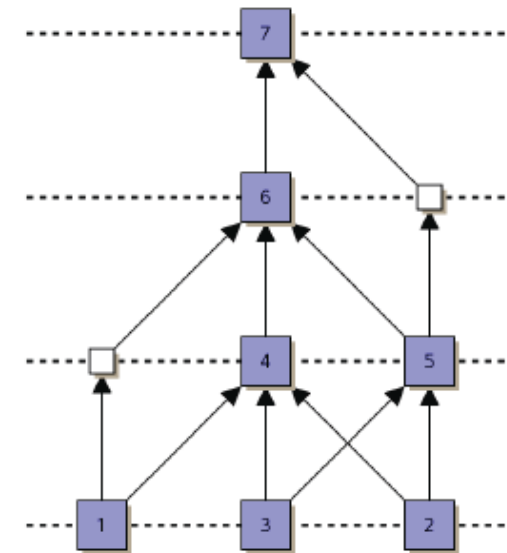


- Kreuzungsreduktion:** Ordne die Knoten in den Schichten um, um die Anzahl der Kantenschnitte zu reduzieren.



4. **Waagrechte Koordinatenzuweisung:** Bestimme die x-Koordinate der Knoten

5. **Wiederherstellung der Zyklen:** Stelle die alte Richtung der Kanten wieder her, die in Schritt 1 invertiert wurden.



Schritt 1: Entfernen von Zyklen

Invertiere die Richtung einiger Kanten, um den Graph azyklisch zu machen.

Problem: Minimale Anzahl von Rückwärtskanten

Gegeben: Ein gerichteter Graph $G = (V, E)$.

Gesucht: Eine Teilmenge R von E kleinster Mächtigkeit, so dass G azyklisch wird, wenn die Kanten von R invertiert werden.

Schwierigkeit: Das Problem ist NP-schwierig.

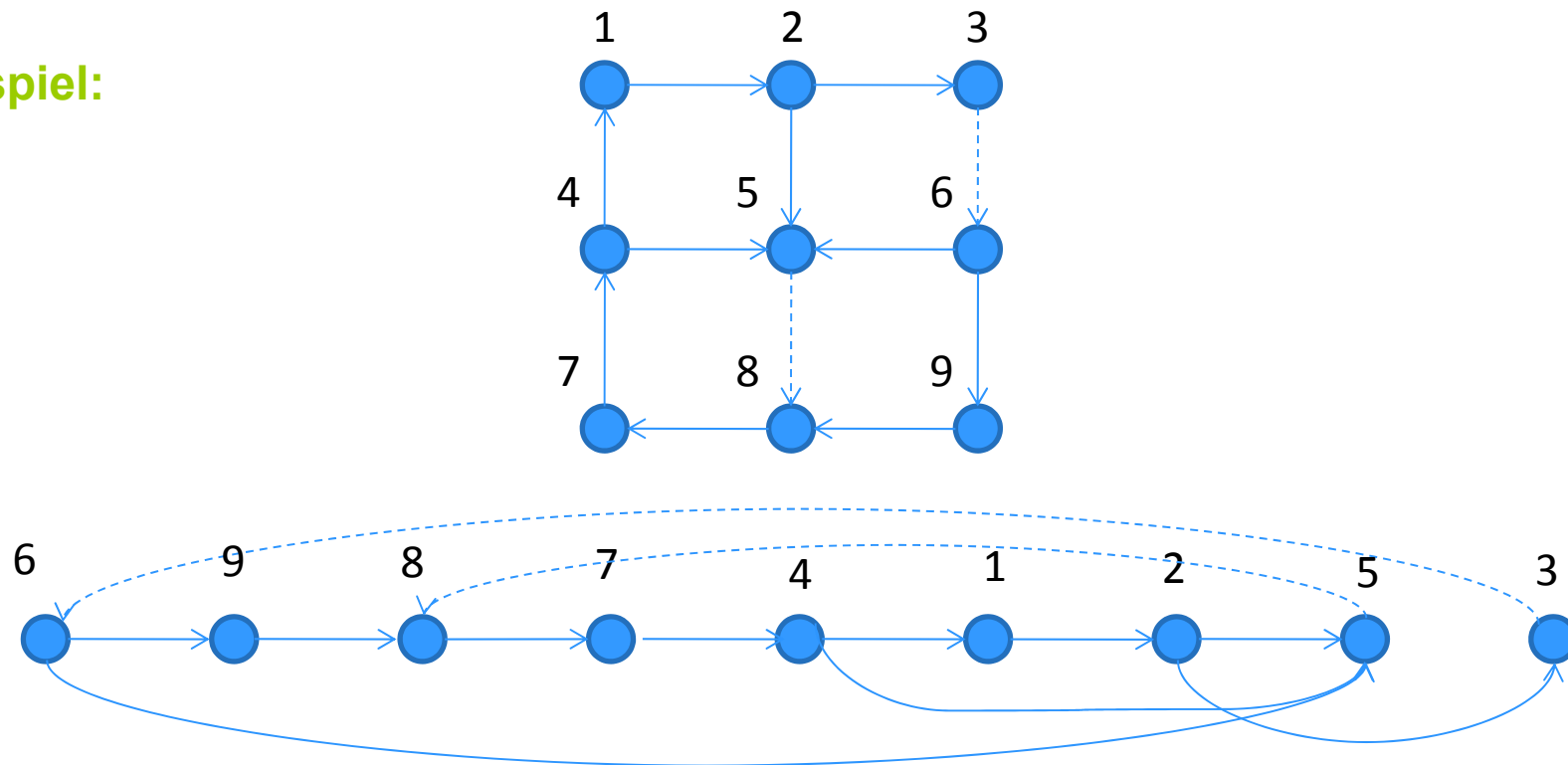
Näherungsweise Lösung: Greedy-Algorithmus zur Zyklentfernung

Greedy-Algorithmus zur Zyklonentfernung

Eingabe: Ein gerichteter Graph $G = (V, E)$.

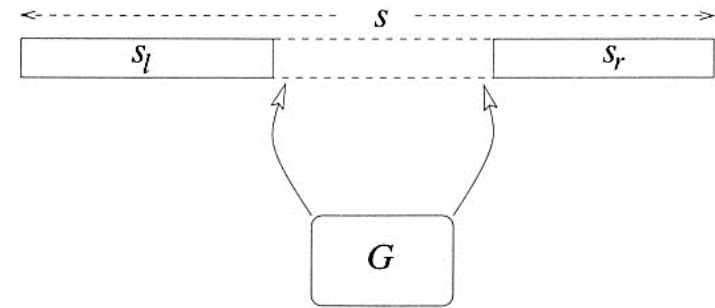
Ausgabe: Eine Anordnung der Knoten, so dass die Anzahl der rückwärts gerichteten Kanten klein ist.

Beispiel:



Ablauf:

1. Initialisiere leere Listen S^l und S^r .
2. Solange G nicht leer ist:
 - a) Solange G eine Senke oder isolierte Knoten enthält, entferne diese aus G und füge sie vorne an S^r an.
 - b) Solange G eine Quelle enthält, entferne diese aus G und füge sie hinten an S^l an.
 - c) Wenn G nicht leer ist, dann
Wähle einen Knoten u , sodass die Differenz zwischen Auswärtsgrad von u minus Einwärtsgrad von u maximal ist, entferne u aus G und füge u an S^l an.
3. Verkette S^l und S^r zu einer Liste S und gib S zurück.



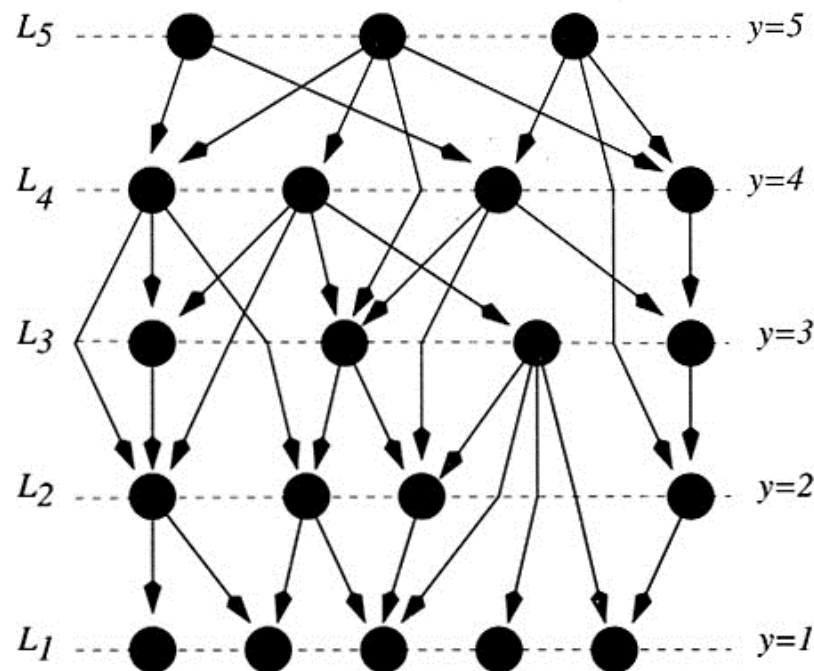
Bemerkungen

1. Es kann gezeigt werden, dass der Algorithmus für einen zusammenhängenden gerichteten Graphen mit n Knoten und m Kanten ohne 2-Zyklen eine Knotenanordnung mit höchstens $m/2 - n/6$ rückwärts gerichteten Kanten erzeugt.
2. Es gibt eine Implementierung des Algorithmus mit linearer Laufzeit.

Schritt 2: Schichtzuordnung:

Weise die Knoten waagrechten Schichten zu (y-Koordinate des Knotens).

- **Algorithmus zur Schichtzuordnung mit längsten Wegen**
- **Algorithmus zur Schichtzuordnung mit beschränkter Breite**
(Coffman-Graham)



*Geschichtete Darstellung
eines gerichteten
azyklischen Graphen.*

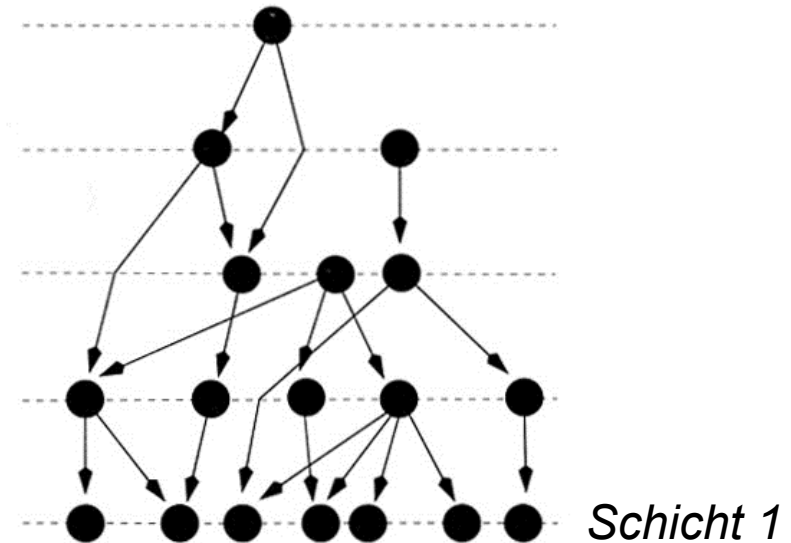
Algorithmus zur Schichtzuordnung mit längsten Wegen

Eingabe: Ein azyklischer gerichteter Graph $G = (V, E)$.

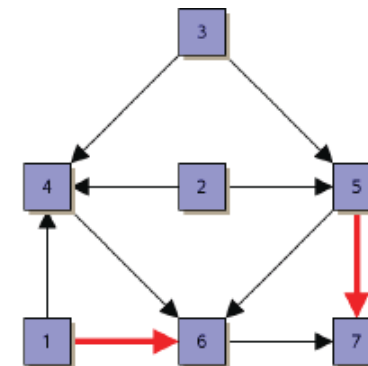
Ausgabe: Eine Schichtzuordnung der Knoten von G .

Ablauf:

1. Platziere alle Senken in Schicht 1.
2. Entferne sukzessive alle Senken aus dem aktuellen Restgraph und ordne sie auf einer neuen Schicht an.



Bemerkung: Dadurch wird ein Knoten v in die Schicht L_{p+1} platziert, wobei p die Länge eines Weges mit größter Kantenanzahl von v zu einer Senke ist.



Eigenschaften des Algorithmus zur Schichtzuordnung mit längsten Wegen:

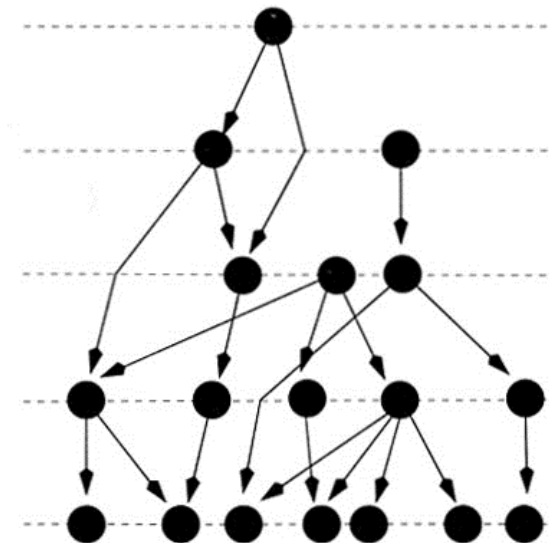
positiv:

- Die Schichtzuordnung kann in linearer Zeit berechnet werden.
- Die Anzahl der Schichten ist minimal.

negativ:

- Die Zeichnung kann recht breit werden.

Beispiel



Algorithmus zur Schichtzuordnung mit beschränkter Breite (Coffman-Graham)

- **Eingabe:** Ein azyklischer gerichteter Graph $G = (V, E)$, eine natürliche Zahl W .
- **Ausgabe:** Eine Schichtzuordnung der Knoten von G mit maximal W Knoten pro Schicht.

Vorgehensweise:

Übernahme eines Verfahrens des Job-Scheduling:

- Bearbeite den Job als nächstes, für den die Bearbeitung der Vorgängerjobs am längsten zurück liegt.
- Zu jedem Zeitpunkt kann nur eine gegebene maximale Anzahl von Jobs parallel bearbeitet werden.

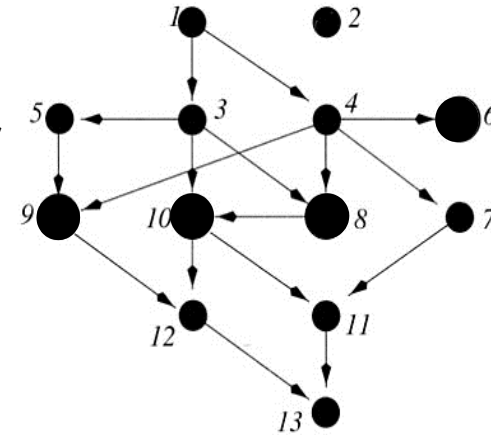
Motivation:

Die Größe der Ordnungsnummer eines Knotens ist in etwa proportional zur Entfernung des Knotens von den Quellen.

Grobablauf des Algorithmus zur Schichtzuordnung mit beschränkter Breite:

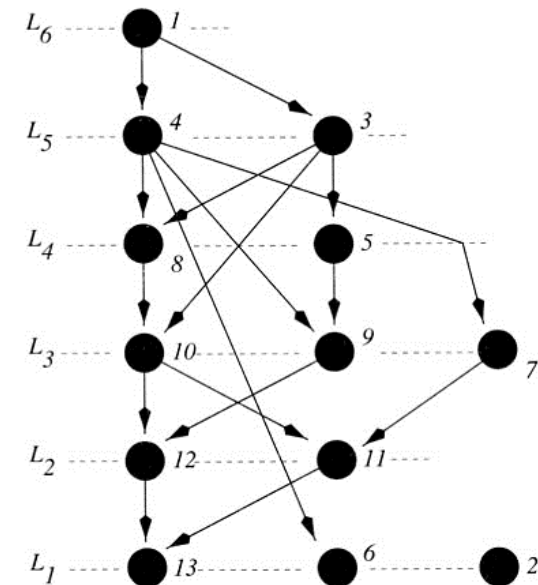
1. Phase - Knotenanordnung:

Weise den jedem Knoten v eine Ordnungsnummer $\pi(v)$ zu. Ordne die Knoten von G durch sukzessives Entfernen von Quellen aus G an, wobei die Quelle v als nächstes gewählt wird, deren Kanten (u, v) die kleinste maximale Ordnungsnummer an den Knoten u hat.



2. Phase - Schichtenaufbau:

Entferne sukzessive Senken von G mit der größten Ordnungsnummer und füge sie in die oberste oder eine neue Schicht ein.



Kompakte Darstellung des Algorithmus:

Require: A reduced directed graph $G = (V, E)$ and a width (integer) W .

Ensure: A layering of G with width W .

All nodes are initially unlabeled.

1. Phase

for $i = 1$ to $|V|$ **do**

Choose an unlabeled node $v \in V$, such that $\{\pi(u) \mid (u, v) \in E\}$ is minimised with respect to \prec „Ordnungskriterium“

$\pi(v) = i$

end for

2. Phase

$k = 1; L_1 = \emptyset; U = \emptyset.$

while $U \neq V$ **do**

$S = \{n \mid n \in V - U, (n, u) \in E, u \in \bigcup_{i=1}^{k-1} L_i\}$

if $S = \emptyset$ or $|L_k| = W$ **then**

$k = k + 1$

else

Choose $u \in S$ such that $\pi(u)$ is maximised.

add u to L_k

add u to U

end if

end while

Ordnungskriterium auf Mengen S und T positiver ganzer Zahlen:

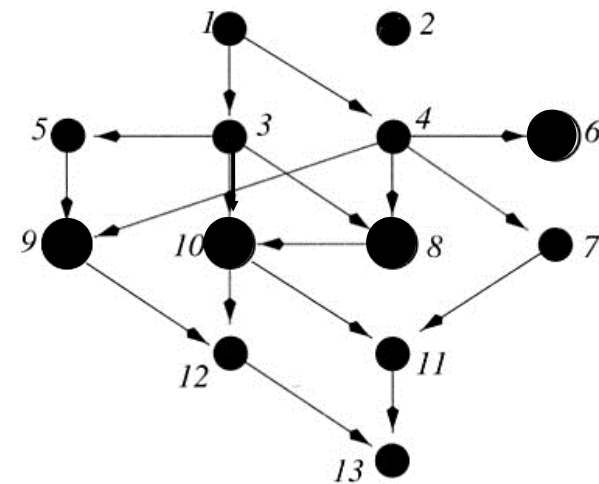
$S < T$ genau dann wenn eine der folgenden Bedingungen gilt:

1. $\emptyset < T$ für alle $T \neq \emptyset$
2. $\max(S) < \max(T)$ für $S \neq \emptyset, T \neq \emptyset$
3. $\max(S) = \max(T)$ und $S - \{\max(S)\} < T - \{\max(T)\}$ für $S \neq \emptyset, T \neq \emptyset$

anschaulich:

lexikographische Ordnung mit dem größten Wert als signifikantester Stelle

$\{1, 4, 7\} < \{3, 8\}$
 $\{3, 4, 9\} < \{1, 5, 9\}$



Beispiel für eine Markierung, die von dem Algorithmus in Schritt 2 erzeugt wird.

Detaillierter Ablauf:

Gegeben: Gerichteter Graph $G(V,E)$ und eine vorgegebene Breite W .

1. Initialisiere alle Knoten als „nicht markiert“.

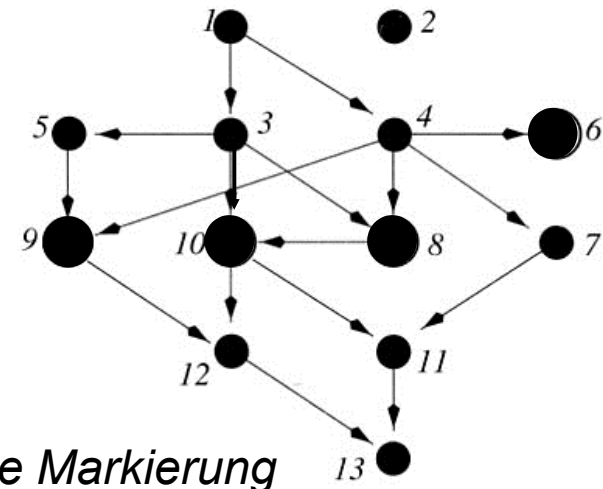
2. Für $i = 1, \dots, |V|$:

{ Wähle einen nicht markierten Knoten v , so dass die Menge
 $\Pi(v) := \{\pi(u) \mid (u, v) \in E\}$ minimal ist bezüglich des
Ordnungskriteriums
 $\pi(v) := i$.
 }

Idee:

d.h. sukzessive Quellen des
 Restgraphen entfernen, deren Kanten
 möglichst weit auf schon bearbeitete
 Knoten zurückverweisen

1. Phase



Beispiel für eine Markierung

3. $k := 1$; $L_1 := \emptyset$; $U := \emptyset$;

4. Solange $U \neq V$ führe aus:

{ Wenn es einen Knoten $u \in V - U$ gibt, so dass

„d.h. u ist eine Senke im Restgraphen und die Auswärtskanten von u zeigen auf schon früher erzeugte Schichten“

- jeder Knoten in $\{v \mid (u, v) \in E\}$ in U ist,
- für jede Kante (u, w) auch $w \in L_1 \cup L_2 \cup \dots \cup L_{k-1}$ gilt }

dann

{ wähle einen solchen Knoten u , für den $\pi(u)$ maximal ist;

„d.h. u verweist auf eine möglichst weit zurückliegende Schicht“

Wenn $|L_k| < W$, dann füge u zu L_k hinzu;

Sonst: $k := k + 1$; $L_k := \{u\}$.

}

Sonst wähle einen Knoten $u \in V - U$ so, dass

„d.h. u verweist auf mind. einen Knoten der aktuell aufzubauenden Schicht“

jeder Knoten in $\{v \mid (u, v) \in E\}$ in U und $\pi(u)$ maximal ist

und setze $k := k + 1$; $L_k := \{u\}$;

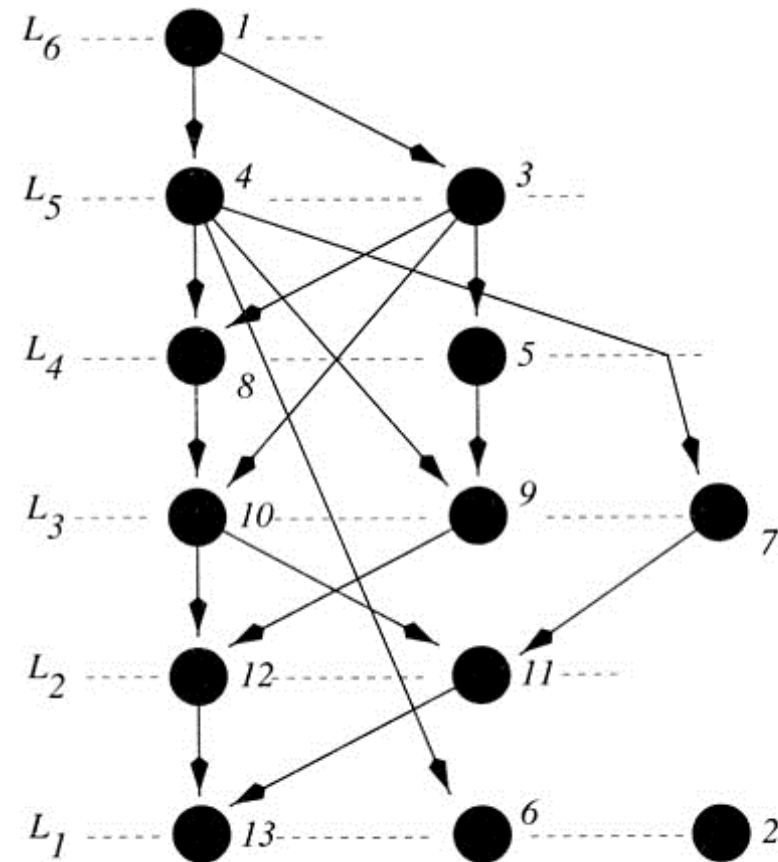
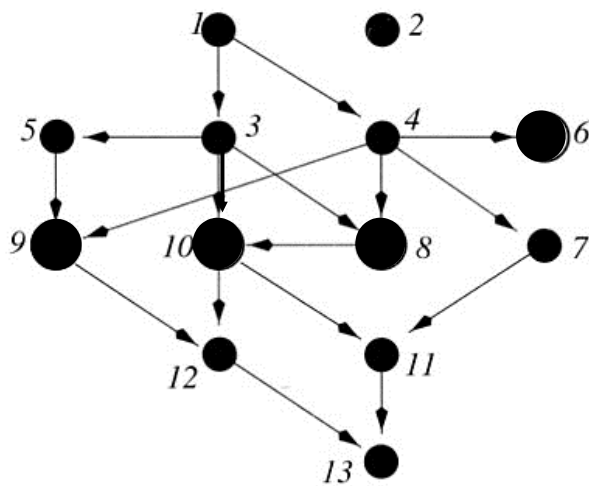
Füge u zu U hinzu;

}

2. Phase

Beispiel:

Schichtzuordnung, die mit dem Coffman-Graham Algorithmus berechnet wurde.



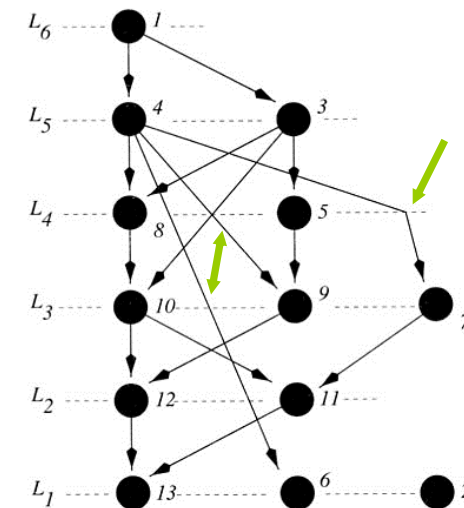
Bemerkungen:

1. Es kann gezeigt werden, dass die Anzahl der Schichten h die Beziehung

$$h \leq \left(2 - \frac{2}{W}\right) h_{\min}$$

erfüllt, wobei h_{\min} die kleinstmögliche Anzahl von Schichten (bestmögliche Lösung) ist.

2. W beschränkt die Anzahl der Knoten auf jeder Schicht. Dabei bleiben die Kanten, die die Schicht möglicherweise durchqueren, außeracht.
3. Es gibt einen Algorithmus, der die Summe der Anzahl der Schichten, die die Kanten durchlaufen, minimiert, siehe [Buch von Battista et al., Kap. 9.1.3](#)



Schritt 3: Kreuzungsreduktion

Ordne die Knoten in den Schichten um, um die Anzahl der Kantenschnitte zu reduzieren.

Bemerkung:

Das Problem der Kreuzungsminimierung ist NP-vollständig.

Verfahrensschema des schichtweisen Abarbeitens:

Wähle eine Anordnung von der ersten Schicht L_1 .

Für $i = 1, \dots, h$:

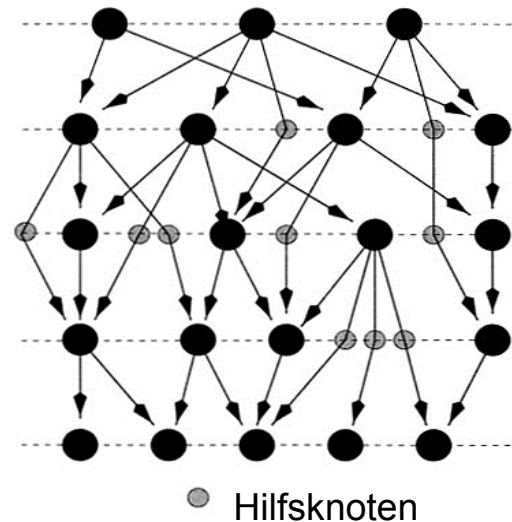
Halte die Anordnung der Knoten auf Schicht L_{i-1} fest und ordne die Knoten auf Schicht L_i so um, dass die Anzahl der Schnittpunkte von Kanten zwischen Schicht L_{i-1} und Schicht L_i reduziert wird.

Problem des schichtweisen Abarbeitens:

Die Kanten zwischen Knoten auf nicht aufeinander folgenden Schichten werden nicht gezielt berücksichtigt.

Lösung:

Einfügen von virtuellen Hilfsknoten (dummy vertices):



Durch das Hinzufügen von Hilfsknoten werden die langen Kanten im geschichteten, gerichteten Graphen geteilt.

Aufgabe bei der Bearbeitung zweier aufeinanderfolgender Schichten:

Zwei-Schichten-Kreuzungsminimierung

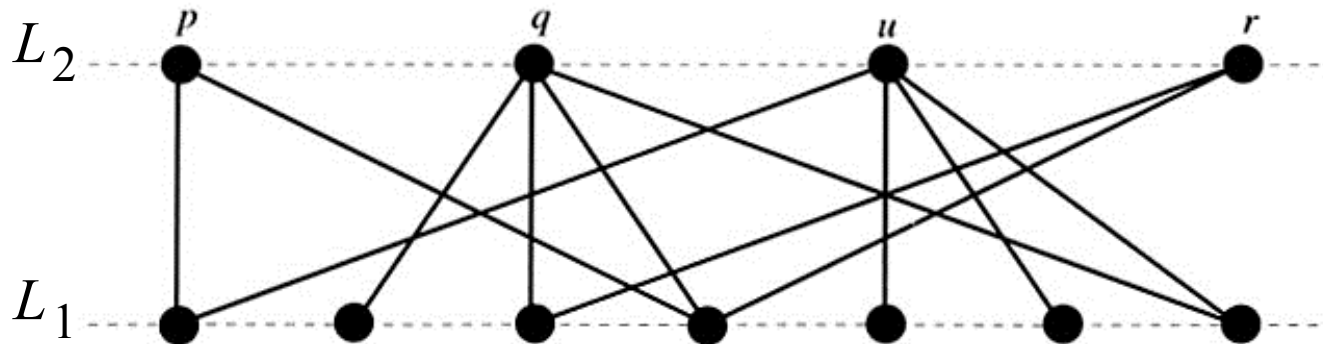
- **Gegeben:** Ein bipartiter Graph $G = (L_1, L_2, E)$ mit einer Knotenanordnung x_1 auf L_1 , d.h. der Zuordnung einer x -Koordinate $x_1(v)$ an jeden Knoten v in L_1 .
- **Gesucht:** Eine Knotenanordnung x_2 auf L_2 , so dass die Anzahl der sich schneidenden Kanten minimal ist.
- **Schwierigkeit:** Das Problem der Zweischichtenminimierung ist NP-vollständig.

Ansätze zur näherungsweise Lösung:

- Sortiermethoden
- Schwerpunkts- und Median-Methode
- Ganzzahlige-Programmierungs-Methoden

Lösung durch eine Sortiermethode

Ordnungsrelation von Knoten: $u \leq p$ gdw. $c_{u,p} \leq c_{p,u}$



$c_{.,.}$	p	q	u	r
p	0	2	1	1
q	5	0	6	3
u	6	9	0	6
r	2	3	2	0

$c_{u,v}$: die Anzahl der Schnittpunkte zwischen Kanten, die zu u inzident sind, und Kanten, die zu v inzident sind, wenn $x_2(u) < x_2(v)$

Algorithmus zur Zwei-Schichten-Kreuzungsminimierung durch Sortieren

Wenn L_2 nicht leer ist, dann

1. Wähle einen Pivot-Knoten $p \in L_2$.

2. $L := \phi$; $R := \phi$;

3. Für jeden Knoten $u \in L_2$, $u \neq p$, führe aus:

Wenn $u \leq p$, dann füge u in L ein, sonst füge u in R ein.

4. Wende Algorithmus rekursiv auf beide bipartiten Graphen an, die durch L und R induziert werden, und füge die beiden Ergebnisanordnungen aneinander.

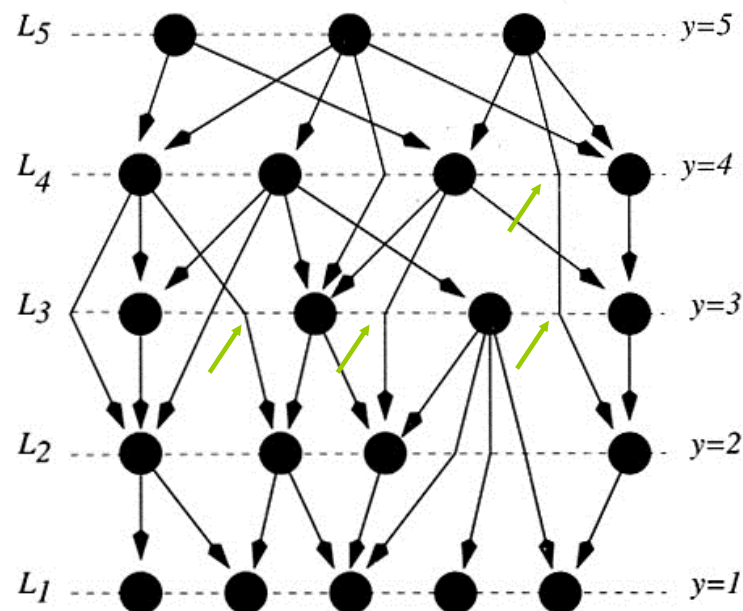
Bemerkung:

- Der Algorithmus zur Zwei-Schichten-Kreuzungsminimierung durch Sortieren ähnelt dem Quicksort-Algorithmus.
- Die Laufzeit ist im schlechtesten Fall quadratisch, im Mittel jedoch $O(n \log n)$, n die Anzahl der Knoten von L_2 .

Schritt 4: Waagrechte Koordinatenzuweisung

Bestimme die x-Koordinate der Knoten.

Ziel: Minimierung der Anzahl der Knicke an den virtuellen Hilfsknoten



Formulierung des Ziels als quadratisches Optimierungsproblem:

Minimiere $\sum_P g(P)$

unter den Nebenbedingungen $x(w) - x(z) \geq \delta$

wobei

- $x(v)$, v Knoten der Schicht, gesucht sind
- die Summation über alle Wege $P = (v_1, \dots, v_k)$ zwischen zwei echten Knoten v_1 und v_k geschieht, die nur Hilfsknoten v_2, \dots, v_{k-1} enthalten (d.h. die Kanten, die Knoten zwischen nicht aufeinanderfolgenden Schichten verbinden),
- $g(P) := \sum_{i=2}^{k-1} (x(v_i) - a_i)^2$
- $a_i = x(v_1) + \frac{i-1}{k-1}(x(v_k) - x(v_1))$ (lineare Interpolation auf der Kante)
- w, z Knoten der Schicht, wobei w rechts von z liegt.

Bemerkungen:

1. Das Problem kann mit Methoden der quadratischen Programmierung gelöst werden.
2. Es gibt Beispiele, bei denen die Lösung mit exponentiellen Flächenbedarf hat. Dieses Problem kann gemindert werden, indem zusätzliche Nebenbedingungen vorgegeben werden, die dafür sorgen, dass die x-Koordinaten innerhalb einer gegebenen Grenze liegen.
3. Eine anderes Ziel könnte es sein, die Kanten möglichst senkrecht zu zeichnen. Dies kann erreicht werden, indem die zu minimierende Zielfunktion durch

$$\sum_{(u,v) \in E} (x(u) - x(v))^2$$

ersetzt wird.

Zusammenfassung

1. **Entfernen von Zyklen:** Invertiere die Richtung einiger Kanten, um den Graph azyklisch zu machen.
2. **Schichtzuordnung:** Weise die Knoten waagrechten Schichten zu (y-Koordinate des Knotens).
3. **Kreuzungsreduktion:** Ordne die Knoten in den Schichten um, um die Anzahl der Kantenschnitte zu reduzieren.
4. **Waagrechte Koordinatenzuweisung:** Bestimme die x-Koordinate der Knoten
5. **Wiederherstellung der Zyklen:** Stelle die alte Richtung der Kanten wieder her, die in Schritt 1 invertiert wurden.

Bibliotheken

- The Open Graph Drawing Framework (OGDF), <http://www.ogdf.net>
- Graphviz - Graph Visualization Software, <http://www.graphviz.org>
- Graph Markup Language (GraphML), <http://graphml.graphdrawing.org/>

D.1. Graphentheorie

- D.1.1. Einleitung
- D.1.2. Gerichtete Graphen
- D.1.3. Ungerichtete Graphen
- D.1.4. Hypergraphen
- D.1.5. Speicherung
- D.1.6. Teilgraphen
- D.1.7. Pfade und Kreise
- D.1.8. Spezielle Eigenschaften

D.2. Zeichnen von Graphen

- D.2.1. Zeichenkonventionen
- D.2.2. Anforderungen
- D.2.3. Vorgehensweisen

D.3. Zeichenstrategien für Graphen

- D.3.1. Topologie-Form-Metrik-Ansatz
 - D.3.1.1. Verfeinerungsebenen
 - D.3.1.2. Algorithmus
 - D.3.1.3. Beispiel
- D.3.2. Hierarchischer Ansatz
 - D.3.2.1. Algorithmus
 - D.3.2.2. Beispiele
 - D.3.2.3. DAG
 - D.3.2.4. Bemerkungen
- D.3.3. Sichtbarkeitsansatz
 - D.3.3.1. Algorithmus
 - D.3.3.2. Beispiel
- D.3.4. Verfeinerungsansatz
 - D.3.4.1. Algorithmus
 - D.3.4.2. Beispiel

D.4. Schichtenweises Zeichnen allg. gerichteter Graphen

- D.4.1. Einleitung
- D.4.2. Algorithmus von Sugiyama
 - D.4.2.1. Übersicht
 - D.4.2.2. Entfernen von Zyklen (Schritt 1)
 - D.4.2.3. Schichtzuordnung (Schritt 2)
 - D.4.2.4. Kreuzungsreduktion (Schritt 3)
 - D.4.2.5. Waagrechte Koordinatenzuweisung (Schritt 4)
 - D.4.2.6. Bemerkungen

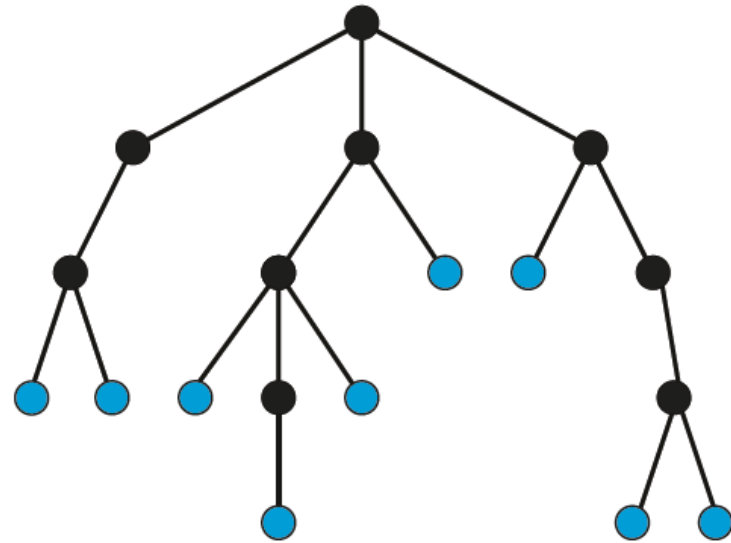
D.5. (Binäre) Bäume

- D.5.1. Konventionen
- D.5.2. Stufenweises Zeichnen
 - D.5.2.1. Einfacher Algorithmus
 - D.5.2.2. Algorithmus von Reingold/Tilford
- D.5.3. Horizontal-vertikal-Zeichnen
 - D.5.3.1. Algorithmus
 - D.5.3.2. Eigenschaften
- D.5.4. Zeichnen durch rekursives Drehen
 - D.5.4.1. Algorithmus
 - D.5.4.2. Eigenschaften
- D.5.5. Freie Bäume
- D.5.6. Bäume mit ausgedehnten Knoten

D.6. Vereinfachung von Polygonzügen

- D.6.1. Einführung
- D.6.2. min - # - Version (Imai, Iri)
- D.6.3. Fehlermaße
- D.6.4. min - e - Version
- D.6.5. Douglas-Peucker-Algorithmus

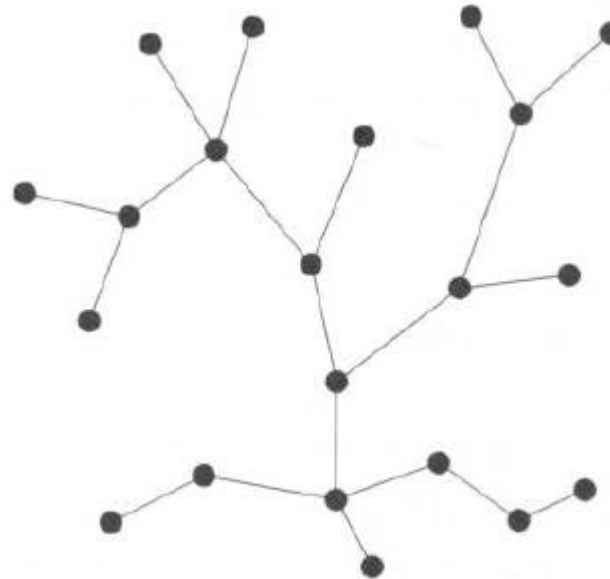
Ein Graph heißt **Baum**, falls er zusammenhängend und kreisfrei ist. Ein Graph ist kreisfrei, wenn er keinen Kreis enthält. Die Knoten vom Grad 1 eines Baums heißen Blätter („blaue Knoten“).



Eigenschaften

- Sei G ein Baum mit mindestens zwei Knoten. Dann besitzt G mindestens einen Knoten vom Grad 1, also mindestens ein Blatt.
- Sei G ein Graph mit n Knoten. Folgende Aussagen sind äquivalent:
 1. G ist ein Baum (d.h. kreisfrei und zusammenhängend).
 2. G ist kreisfrei und hat $n - 1$ Kanten.
 3. G ist zusammenhängend und hat $n - 1$ Kanten.

- Ein Graph, der keinen Kreis enthält, ist ein **Baum**.
- Die Knoten vom Grad 1 eines Baumes sind seine **Blätter** (ausgenommen der Wurzelknoten).
- Ein zusammenhängender Graph von Bäumen ist ein **Wald**. Ein Wald ist somit ein Graph, dessen Komponenten Bäume sind.
- Ein Teilgraph von G , der jeden Knoten aus V enthält und ein Baum ist, heißt **Spannbaum** zu G .

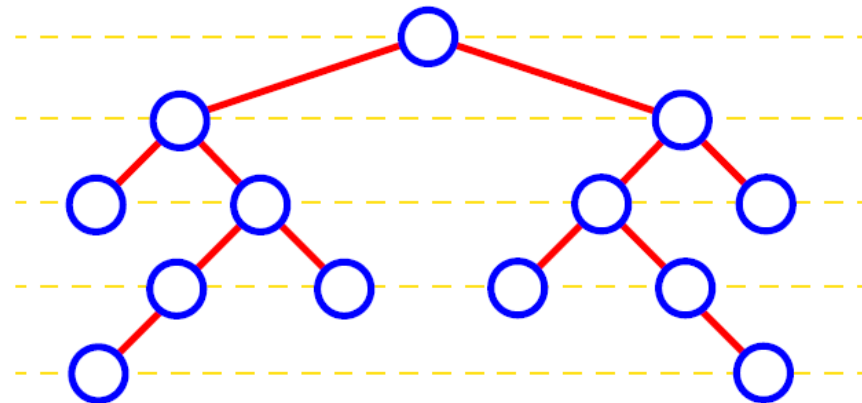


Konventionen

- planar, geradlinige Kanten, aufwärts gerichtet (Eltern über den Kindern)
- Minimierung der Zeichenfläche, Berücksichtigung von Symmetrien und isomorphen Unterbäumen
- stufenweises Zeichnen: Knoten mit demselben Abstand von der Wurzel sollten auf einer gemeinsamen waagrechten Geraden liegen.

Bemerkung:

Das stufenweise Zeichnen kann zu $\Omega(n^2)$ Zeichenflächenbedarf führen, wobei n die Anzahl der Knoten ist.



In-Order Traversierung:

- Durchlaufe den linken Unterbaum.
- Besuche Wurzel R.
- Durchlaufe den rechten Unterbaum

Reihenfolge: C D B E A G H F

Pre-Order Traversierung:

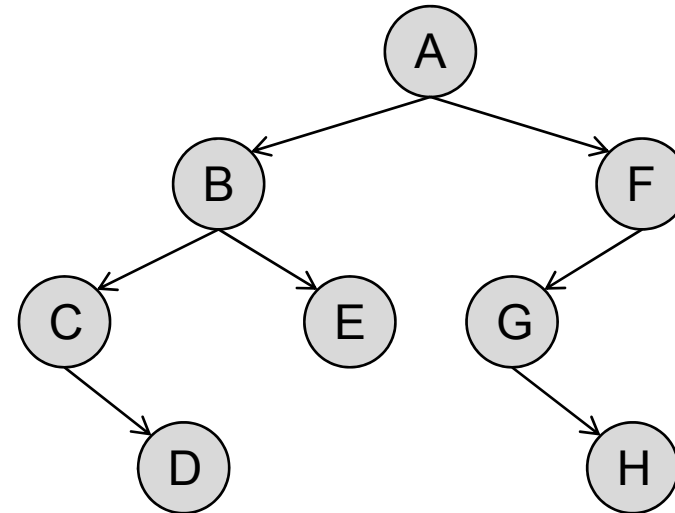
- Besuche Wurzel R.
- Durchlaufe den linken Unterbaum.
- Durchlaufe den rechten Unterbaum.

Reihenfolge: A B C D E F G H

Post-Order Reihenfolge:

- Durchlaufe den linken Unterbaum.
- Durchlaufe den rechten Unterbaum.
- Besuche Wurzel R.

Reihenfolge: D C E B H G F A



Breitensuche:

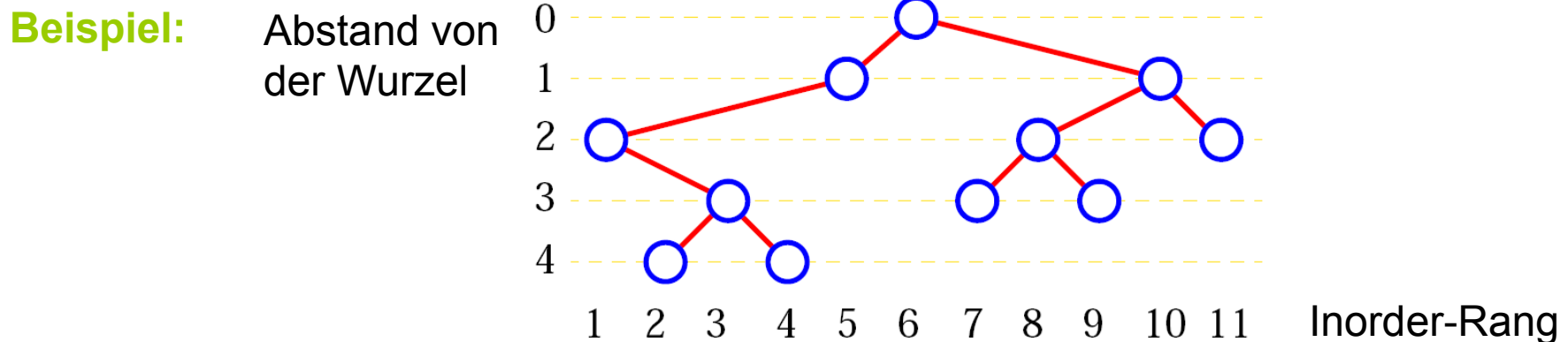
- Besuche Wurzel R
- Durchlaufe die Ebenen von links nach rechts

Reihenfolge: A B F C E G D H

Einfacher Algorithmus zum stufenweisen Zeichnen von Binärbäumen

Platziere den Baumknoten v an den Gitterpunkt $(x(v), y(v))$, wobei

- $x(v)$ der Index von v beim **Inorder**-Durchlauf des Baums
- $y(v)$ der Abstand von v von der Wurzel des Baums.



Eigenschaften des Algorithmus

positiv: Ein Elternknoten ist stets zwischen seinem linken und rechten Kind

- negativ:**
- Ein Elternknoten ist nicht immer zwischen den Kindern zentriert gezeichnet.
 - Die Breite der Zeichnung ist proportional zu Anzahl der Knoten.

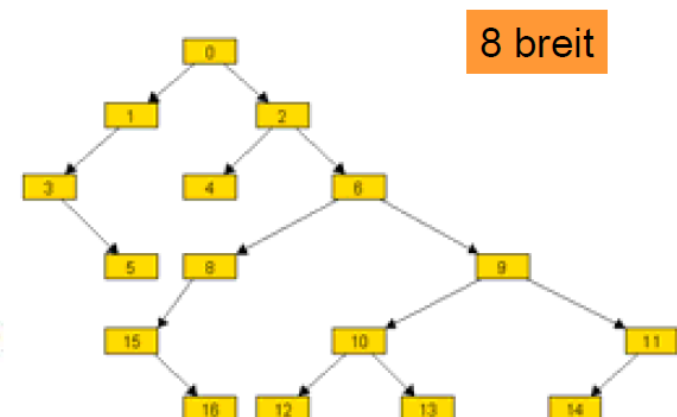
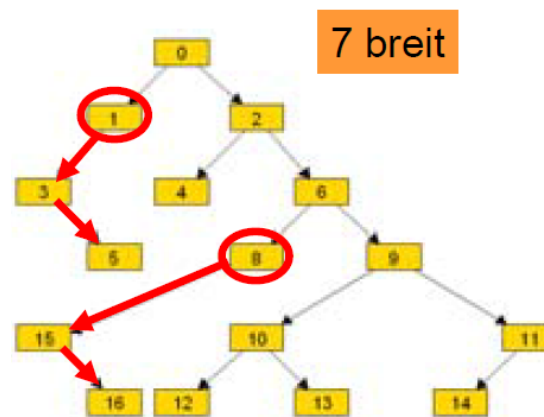
Ästhetikkriterien für Binärbäume

1. „Schichtenzeichnungen“, d.h. Knoten mit gleicher Entfernung zur Wurzel erhalten gleiche Schicht (y-Koordinaten)
2. Linker Kindknoten soll links vom Elternknoten liegen, rechter Kindknoten soll rechts vom Elternknoten liegen
3. Elternknoten sollte zentriert über den Knoten der Kindern liegen
4. Zwei isomorphe Unterbäume sind gleich gezeichnet
5. Ein Baum und sein Spiegelbild erhalten spiegelbildliche Zeichnungen

Ergänzende Ziele:

- Gitterzeichnung, d.h. ganzzahlige Koordinaten
- Bestimme die x-Koordinaten, sodass Weite minimal

Aesthetikkriterium 4. (rechts)
widerspricht sich mit der
Forderung der minimalen
Weite (links)



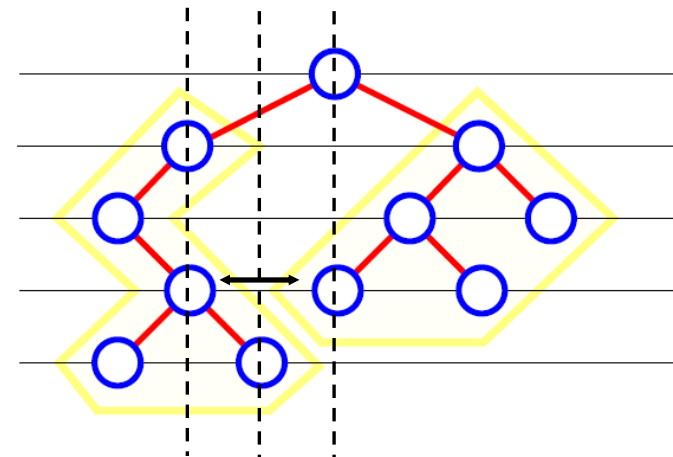
Algorithmus von Reingold und Tilford

- Stufenweises Zeichnen von Binärbäumen
- rekursiver Algorithmus (**Postorder**-Traversierung):

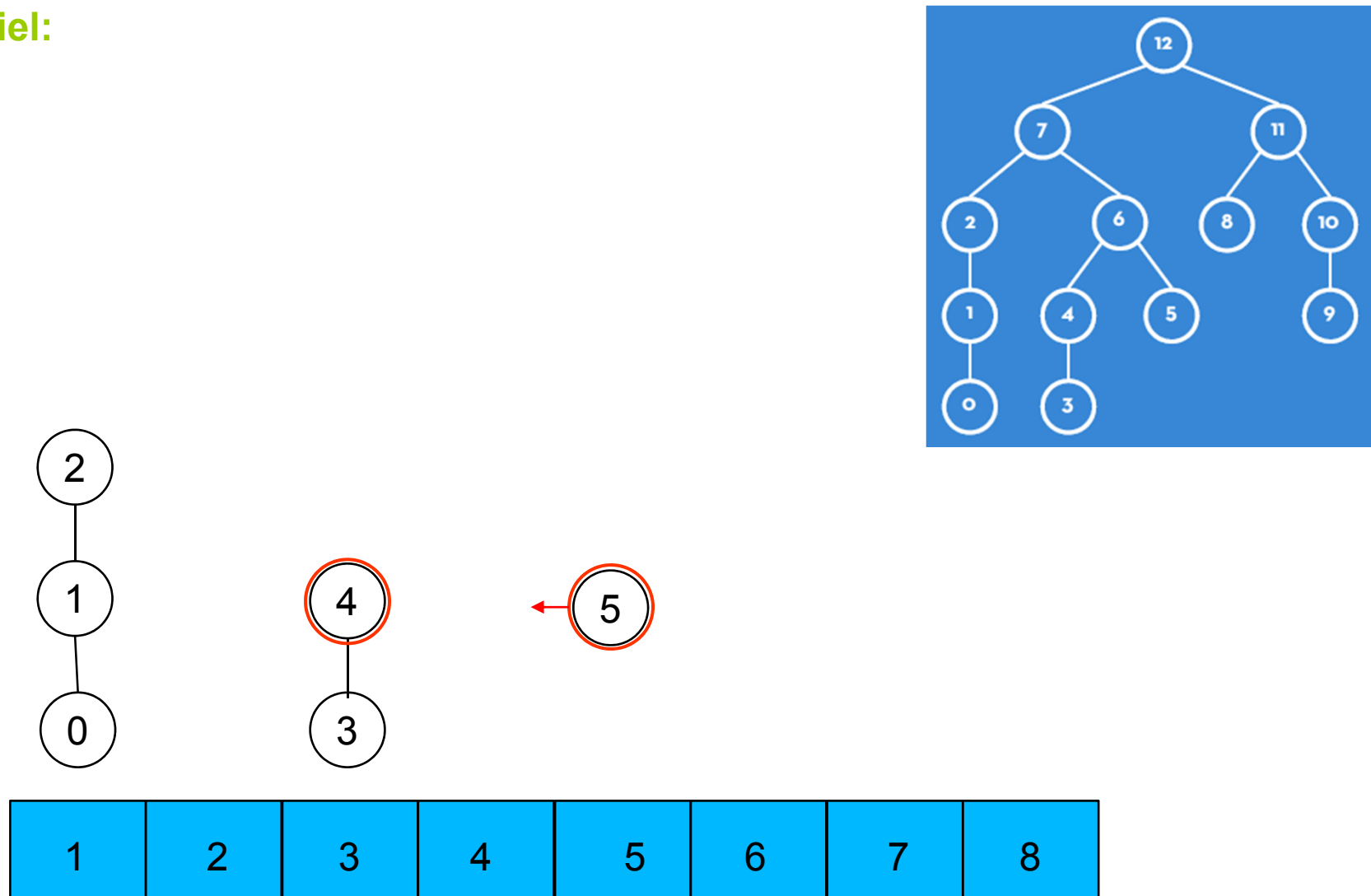
Falls der Baum nur einen Knoten, zeichne den Knoten

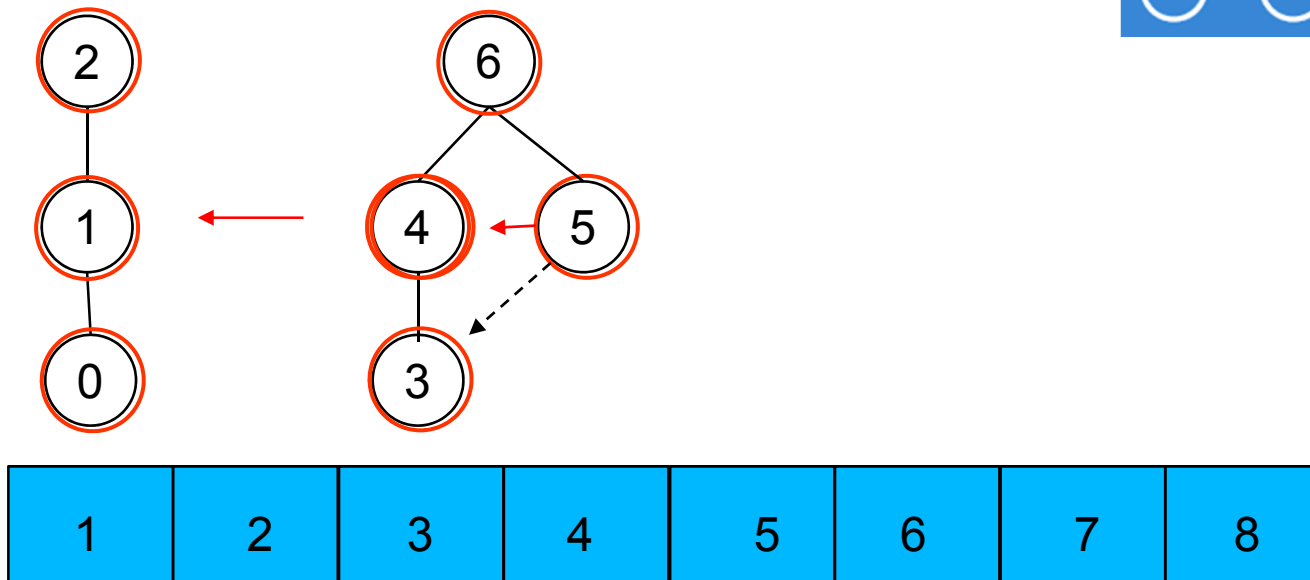
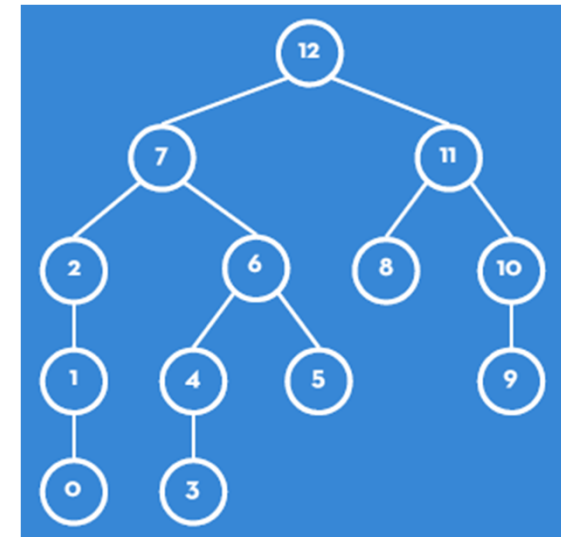
Sonst:

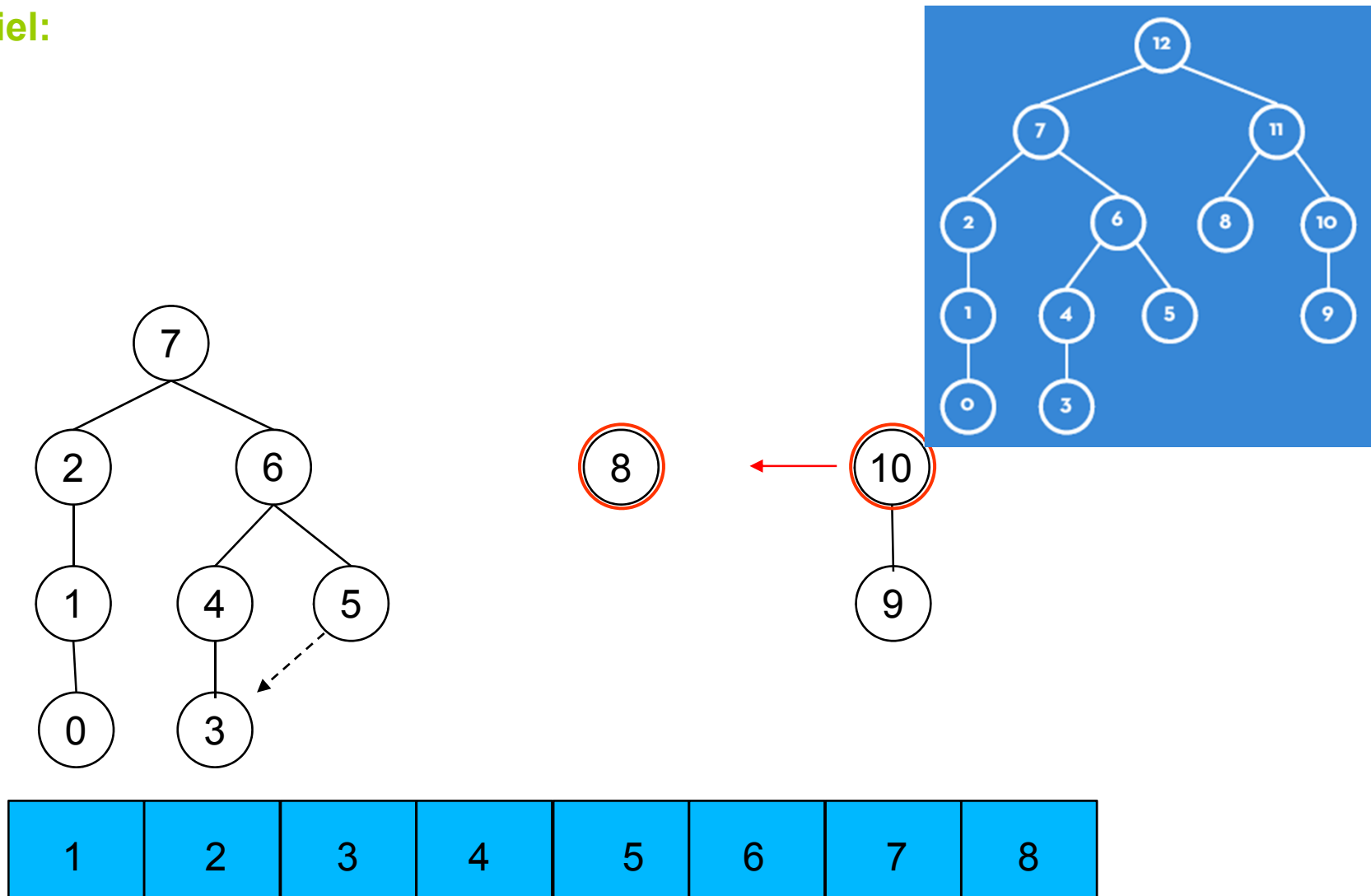
1. Wende den Algorithmus auf den linken und den rechten Unterbaum rekursiv an.
2. Wenn es zwei Kinder gibt dann
 - a) Platziere die beiden erhaltenen Zeichnungen der Unterbäume in waagrechtem Abstand 2.
 - b) Platziere die Wurzel des Baums eine Stufe darüber und in der Mitte zwischen den Kindern.
3. Sonst:
platziere die Wurzel in waagrechtem Abstand 1 vom Kind.

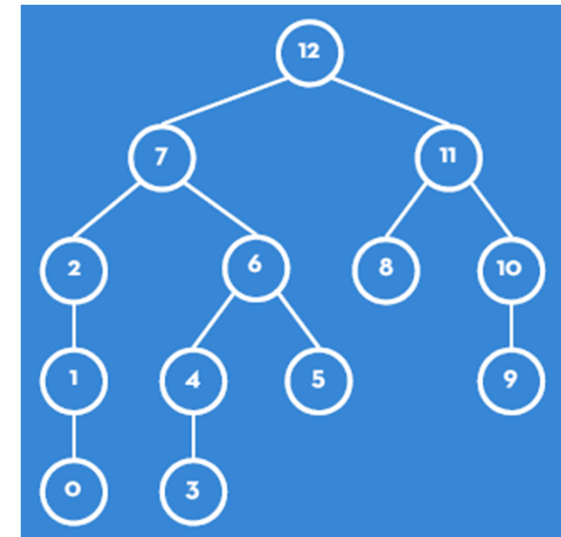
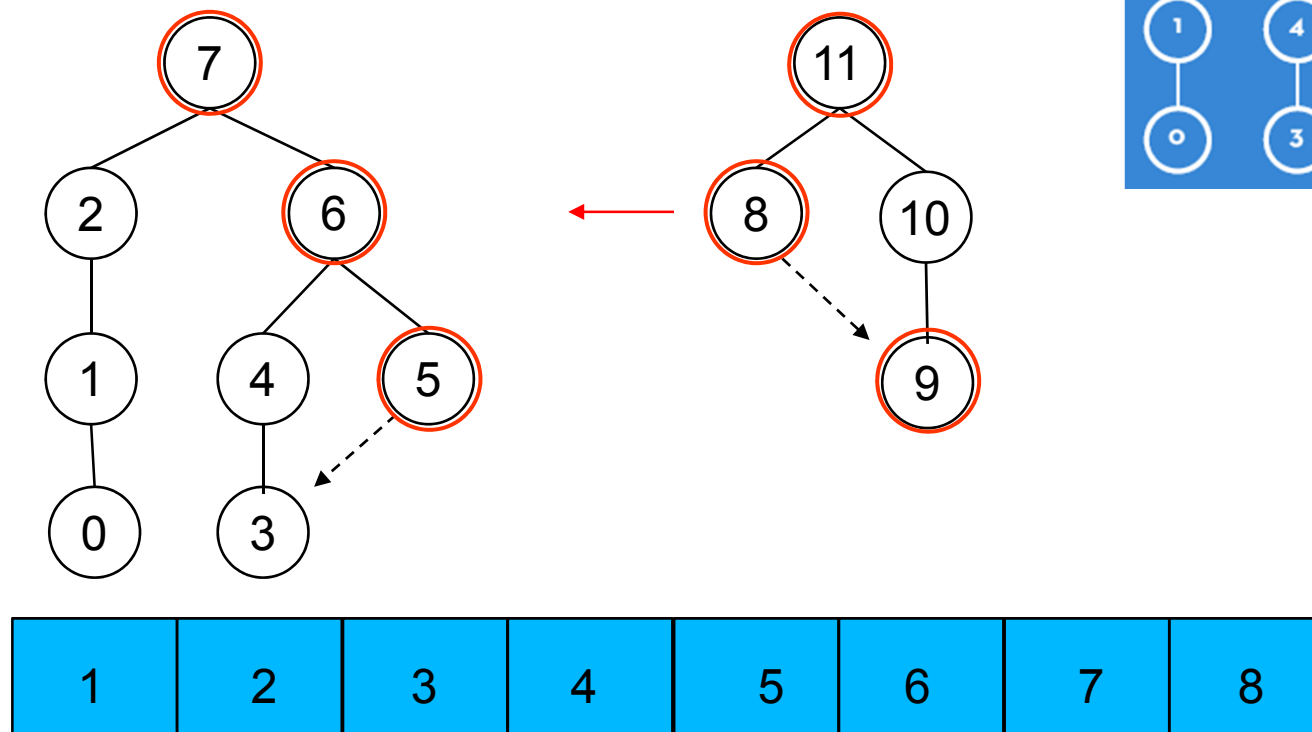


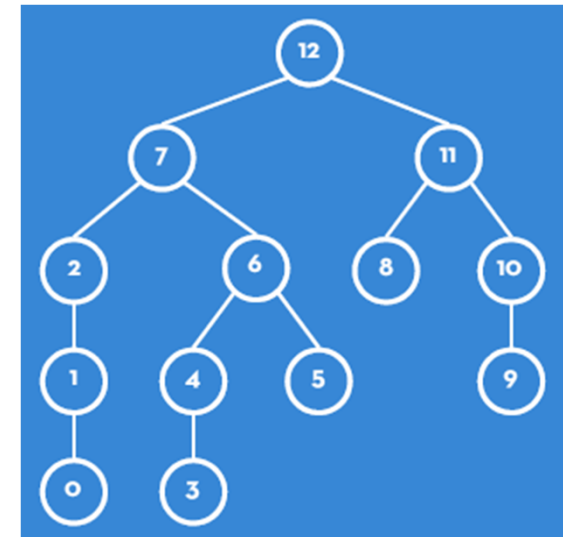
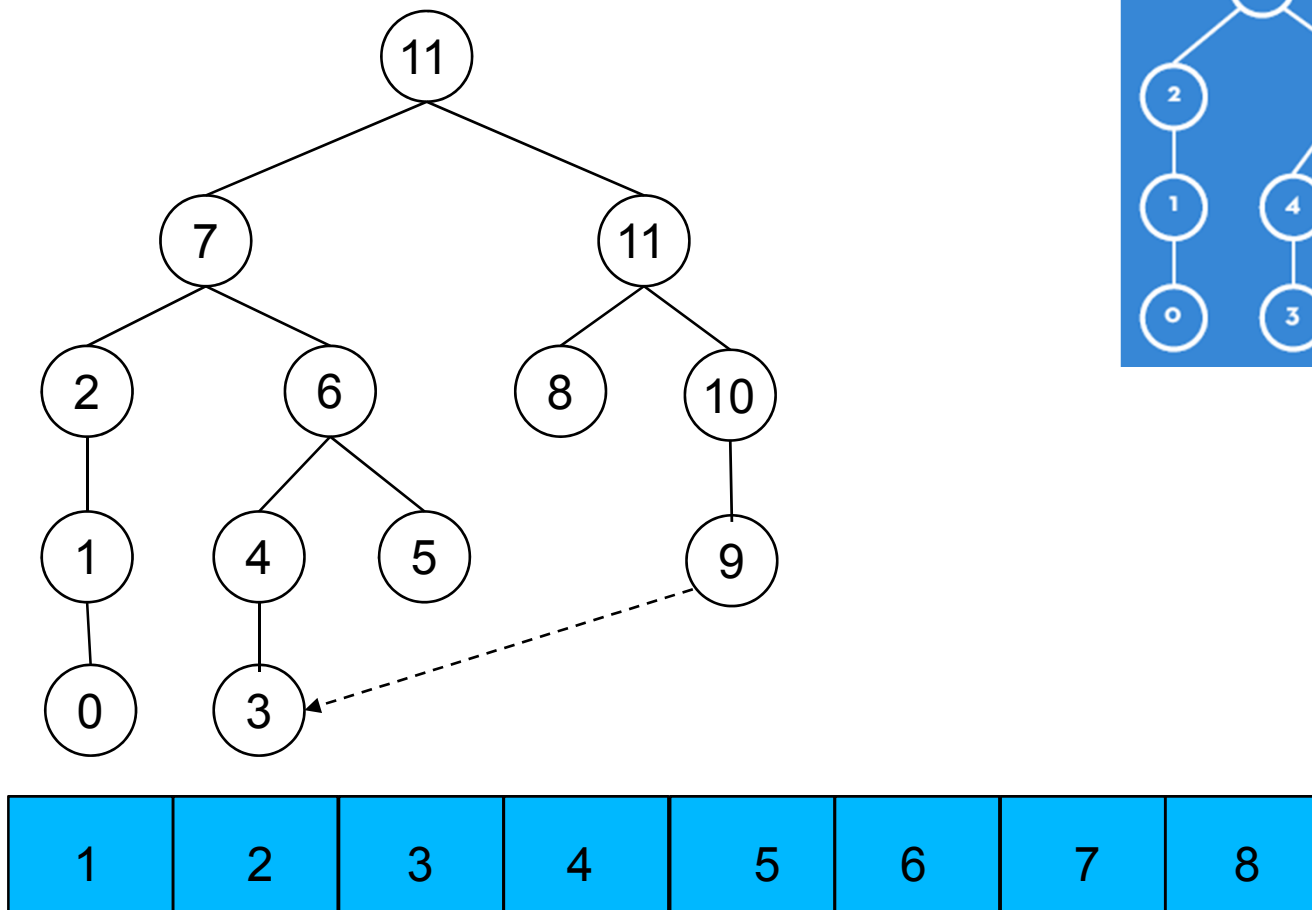
Beispiel:



Beispiel:

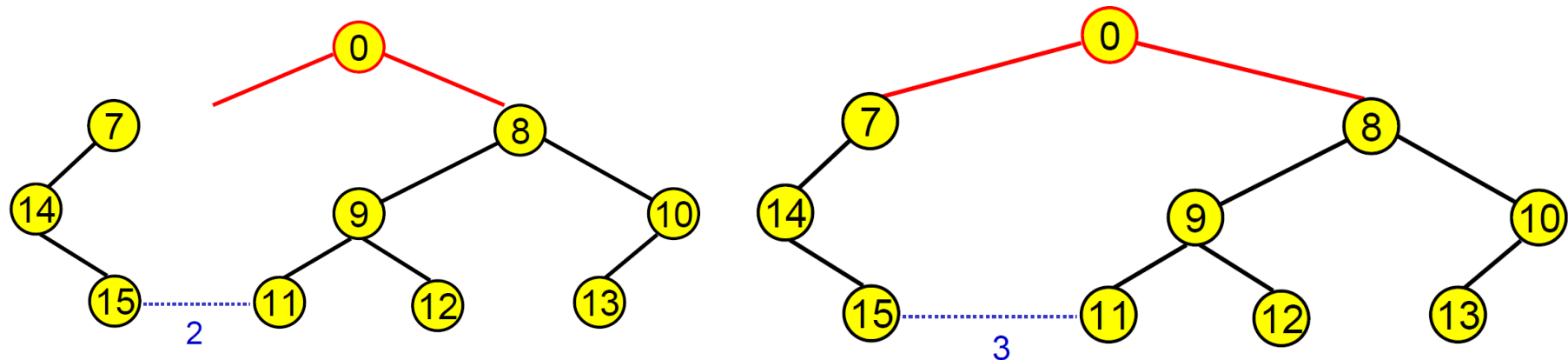
Beispiel:

Beispiel:

Beispiel:

Problem: ungerader Abstand \rightarrow nicht-ganzzahlige Koordinate

Erste Idee: rücke rechten UB um 1 weiter nach rechts



Resultierendes Problem: Weite größer als notwendig

Zweite Idee: Lösung A in Verbindung mit separiere mit Distanz 1

Erweiterte Variante Algorithmus von Reingold/Tilford

Falls der Baum nur einen Knoten, zeichne den Knoten

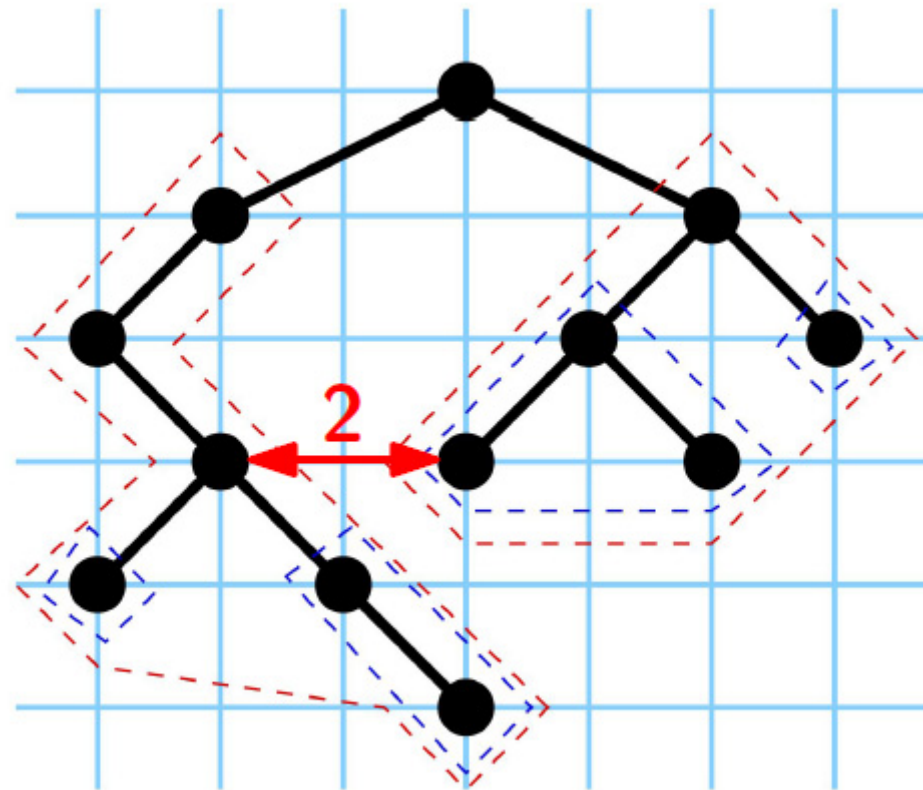
Sonst:

1. Wende den Algorithmus auf den linken und den rechten Unterbaum rekursiv an.
2. Wenn es zwei Kinder gibt dann
 - a) Platziere die beiden erhaltenen Zeichnungen der Unterbäume in waagrechtem Abstand **1**.
 - b) Platziere die Wurzel des Baums eine Stufe darüber und in der Mitte zwischen den Kindern.
 - c) falls der Abstand ungerade, dann schiebe rechten UB um 1 weiter nach rechts
3. Sonst:
platziere die Wurzel in waagrechtem Abstand 1 vom Kind.

Erweiterte Variante Algorithmus von Reingold/Tilford

Zwei Phasen:

1. postorder (bottom-up):
Konturen und x-Offsets zum
Vorgänger bestimmen
2. preorder (top-down):
absolute Koordinaten berechnen



Erweiterte Variante Algorithmus von Reingold/Tilford

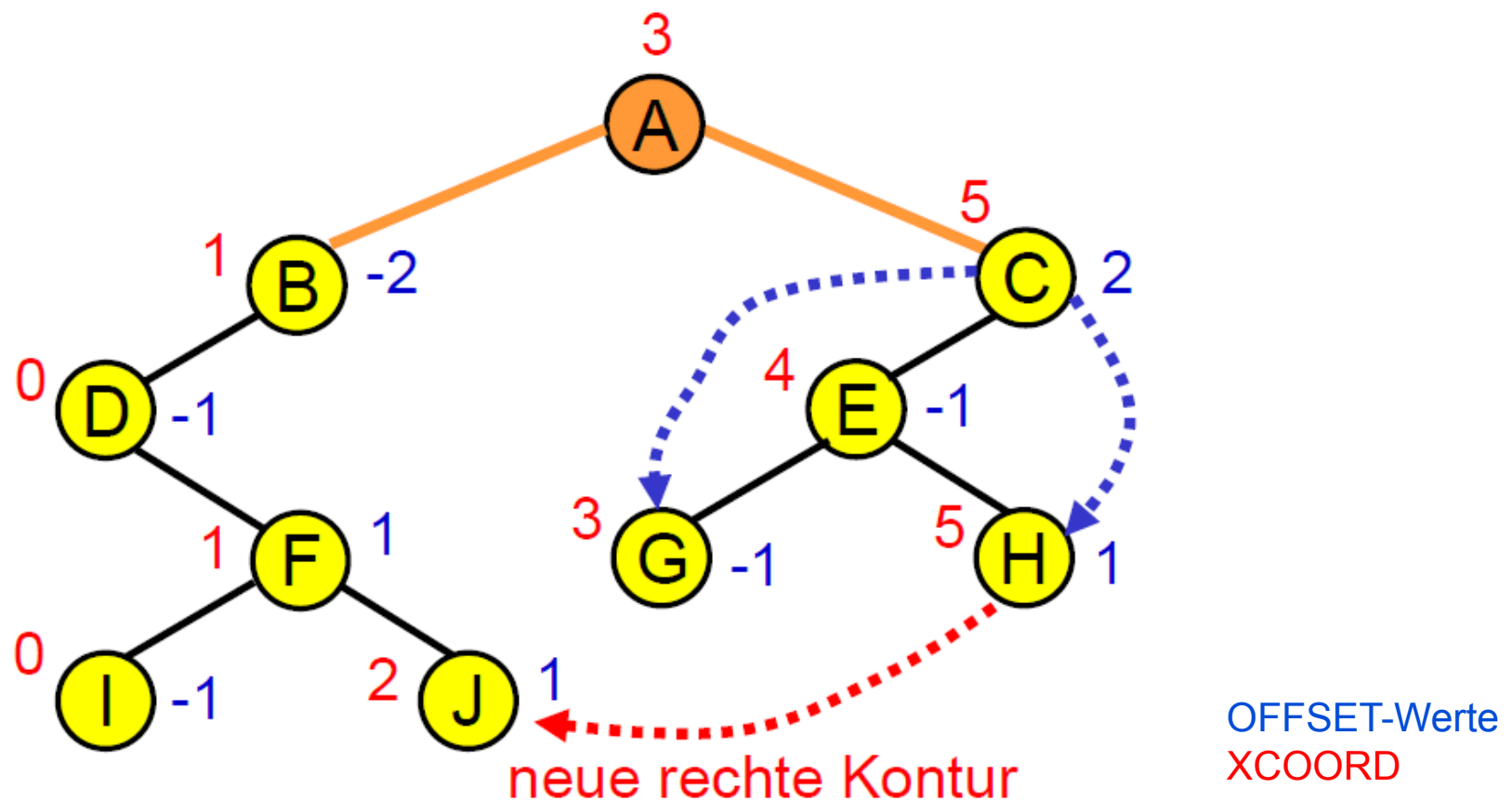
- (1) Durchlaufe den aktuellen (Teil-)Baum T in postorder:
- (2) Initialisiere: $MINDIST=1$
- (3) Platziere die Wurzeln der UB in $MINDIST$ zueinander
- (4) Wandere entlang der rechten Kontur von T_L und der linken Kontur von T_R nach unten und erhöhe dabei $MINDIST$, falls notwendig
- (5) Setze $OFFSET(T_L) = \text{round}(-(MINDIST+1)/2)$
- (6) Setze $OFFSET(T_R) = \text{round}((MINDIST+1)/2)$
- (7) Bei ungleicher Tiefe von T_L und T_R : Bestimme neue Konturen von T (setze Link zum Rand des tieferen)
- (8) Laufe durch linken Rand um absoluten $OFFSET$ von Wurzel W zu erhalten $\rightarrow XCOORD(W)$
- (9) Durchlaufe T in Preorder-Sequenz und lege die x-Koordinaten aller Knoten fest

1. Phase

2. Phase

Erweiterte Variante Algorithmus von Reingold/Tilford

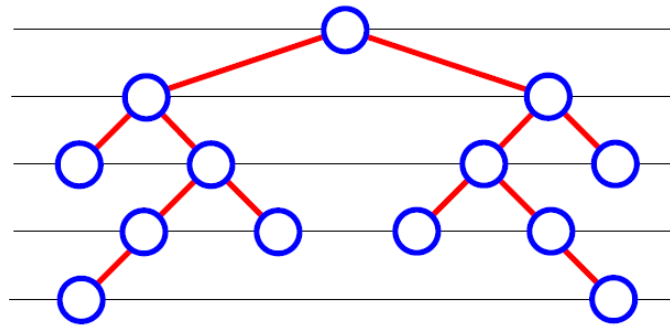
Beispiel:



Eigenschaften des rekursiver Algorithmus

Positiv:

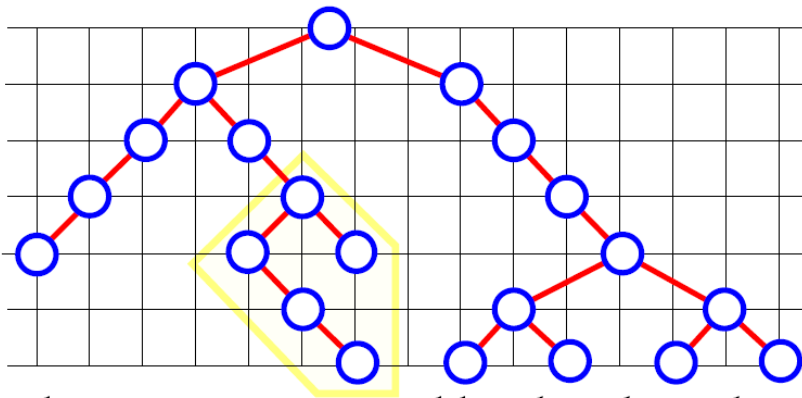
- Die Elternknoten liegen in der Mitte zwischen ihren Kindern.



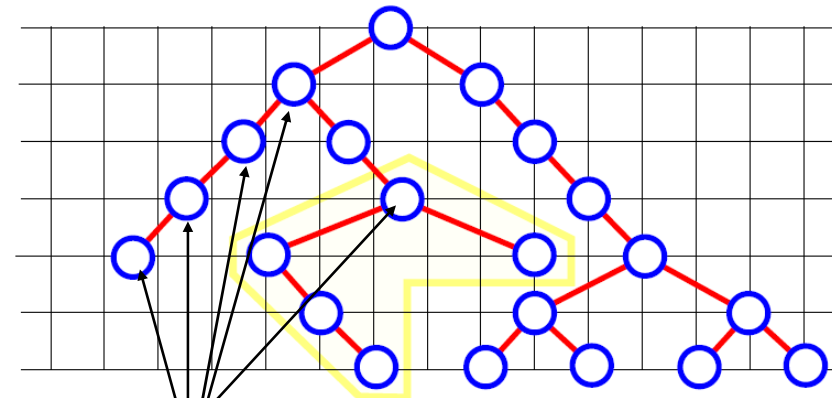
- Baum hat eine „geringe“ Breite.
- Algorithmus kann so implementiert werden, dass er lineare Laufzeit hat
- Algorithmus kann auf nicht binäre Bäume erweitert werden.

Negativ: Der Algorithmus liefert nicht die minimal mögliche Breite.

Ergebnis des Algorithmus



Zeichnung mit geringerer Breite



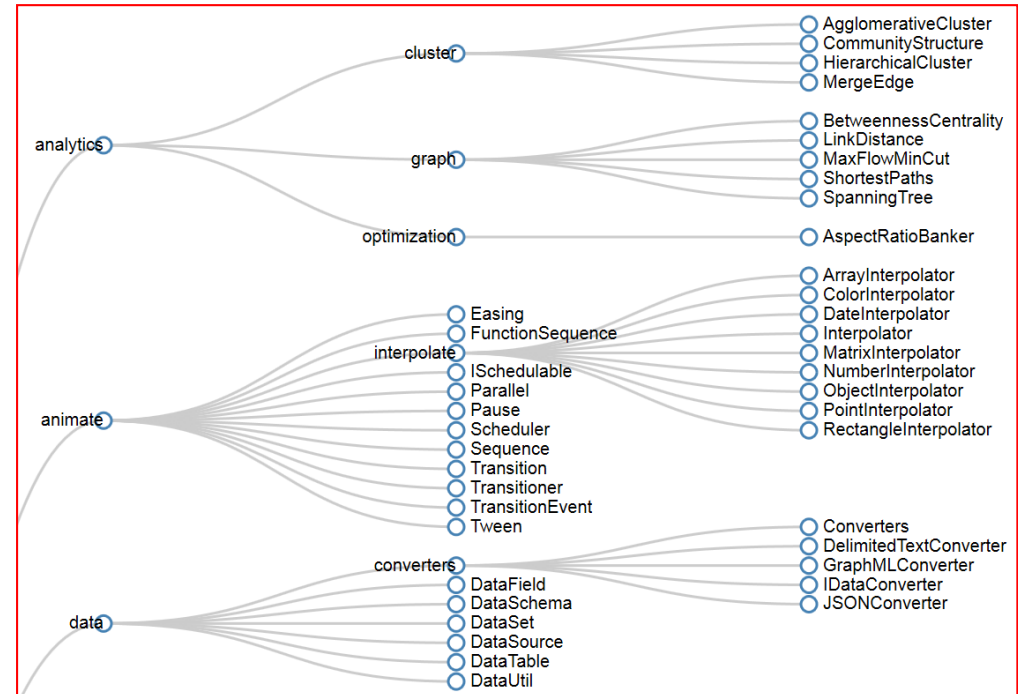
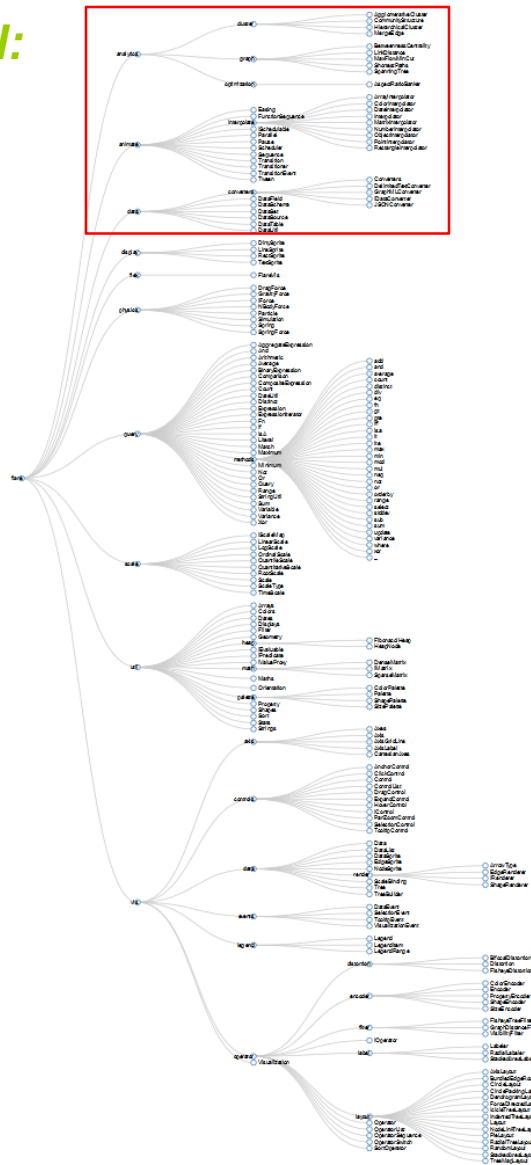
auch Zwischenkoordinaten
zugelassen

Bemerkung: Falls die Koordinaten der Knoten ganzzahlig sein sollen, ist die Minimierung der Breite NP-schwierig.

Weitere Details:

- Buch „Graph Drawing: Algorithms for the Visualization of Graphs“, G. Battista, P. Eades, R. Tamassia, I. Tollis, Prentice Hall, 1998
- Vorlesung „Automatisches Zeichnen von Graphen“, Prof. Dr. P. Mutzel, LS11, TU Dortmund, Informatik

Beispiel:




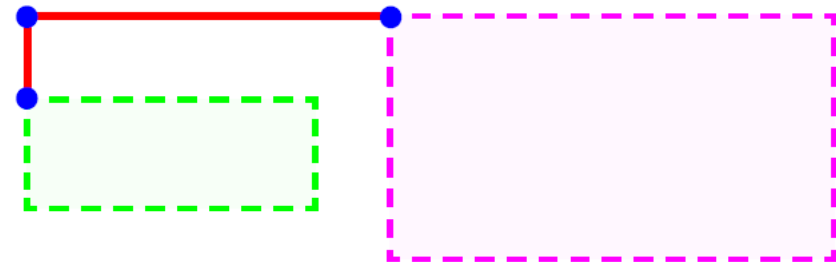
Klassenhierarchie

Algorithmus zum horizontal-vertikal-Zeichnen von Binärbäumen

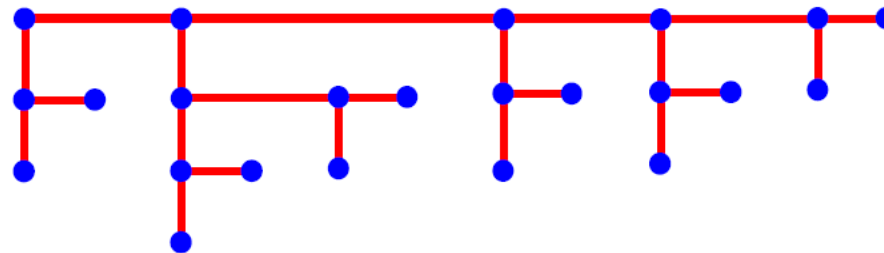
Falls der Baum nur einen Knoten, zeichne den Knoten

Sonst:

1. Wende den Algorithmus auf den linken und den rechten Unterbaum rekursiv an.
 2. Zeichne den Teilbaum mit der größeren Anzahl Knoten rechts und den Teilbaum mit der kleineren Anzahl Knoten unterhalb der Wurzel.
- 



Beispiel:

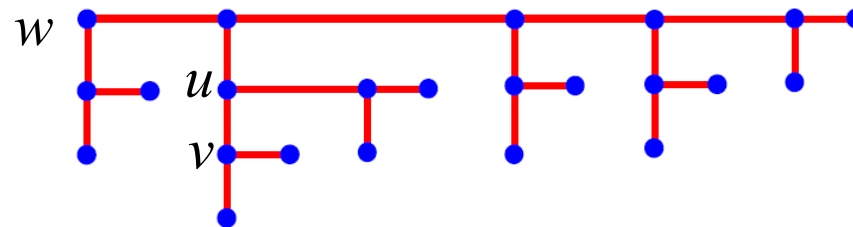


Eigenschaften des Algorithmus zum horizontal-vertikal-Zeichnen

- Die Zeichnung hat $O(n \log n)$ Fläche, $O(n)$ Breite und $O(\log n)$ Höhd.

Begründung zur Höhe:

- Alle senkrechten Kanten haben die Länge 1.
 - Die Höhe der Graphik ist gleich der Länge des Wegs (Anzahl senkrechter Kanten) von der Wurzel w zu einem untersten Knoten l in der Zeichnung.
 - Sei (u, v) eine senkrechte Kante auf dem Weg von w zu l . Dann ist der Unterbaum mit Wurzel u mindestens doppelt so groß wie der Unterbaum mit Wurzel v . Damit enthält der Weg von w zu l höchstens $\log n$ senkrechte Kanten.
- Der Algorithmus kann auf nicht binäre Bäume erweitert werden.



Algorithmen zum Zeichnen von Binärbäumen durch rekursives Drehen

Sei n die Anzahl der Knoten eines gegebenen Baums, $n(v)$ die Anzahl der Knoten des Unterbaums mit Wurzel v , v ein Knoten des gegebenen Baums, $\text{links}(v)$ beziehungsweise $\text{rechts}(v)$ das linke bzw. rechte Kind von v .

Ablauf:

1. Für jeden Knoten u : vertausche seine Kinder so, dass $n(\text{links}(u))$ kleiner oder gleich $n(\text{rechts}(u))$.
2. Finde ersten Knoten v auf dem am weitesten rechts verlaufenden Weg, für den gilt:

$$n(\text{rechts}(v)) \leq n - (n \log n)^{1/2} < n(v).$$
3. Zeichne die linken Unterbäume auf dem Weg von der Wurzel nach v mit dem Algorithmus zum horizontal-vertikal Zeichnen.
4. Zeichne die Unterbäume T' und T'' rekursiv.

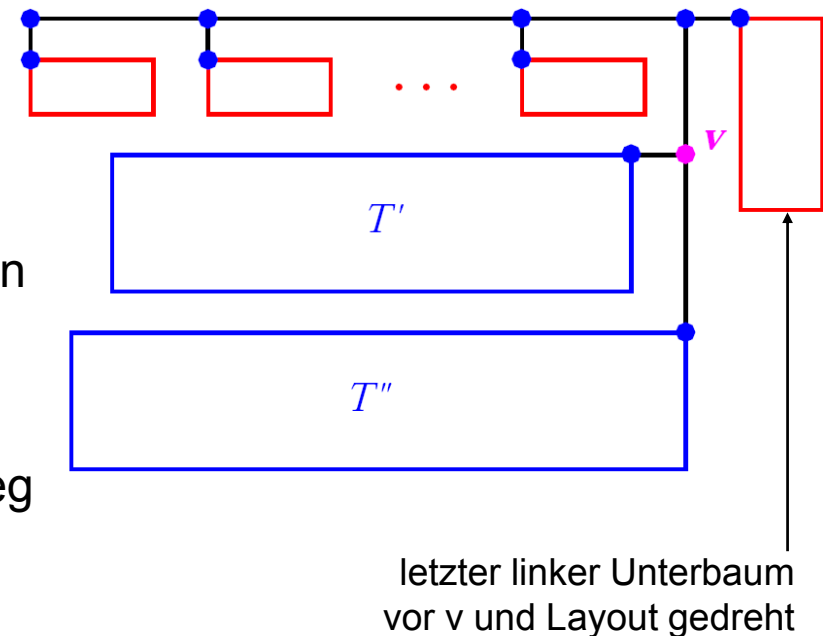
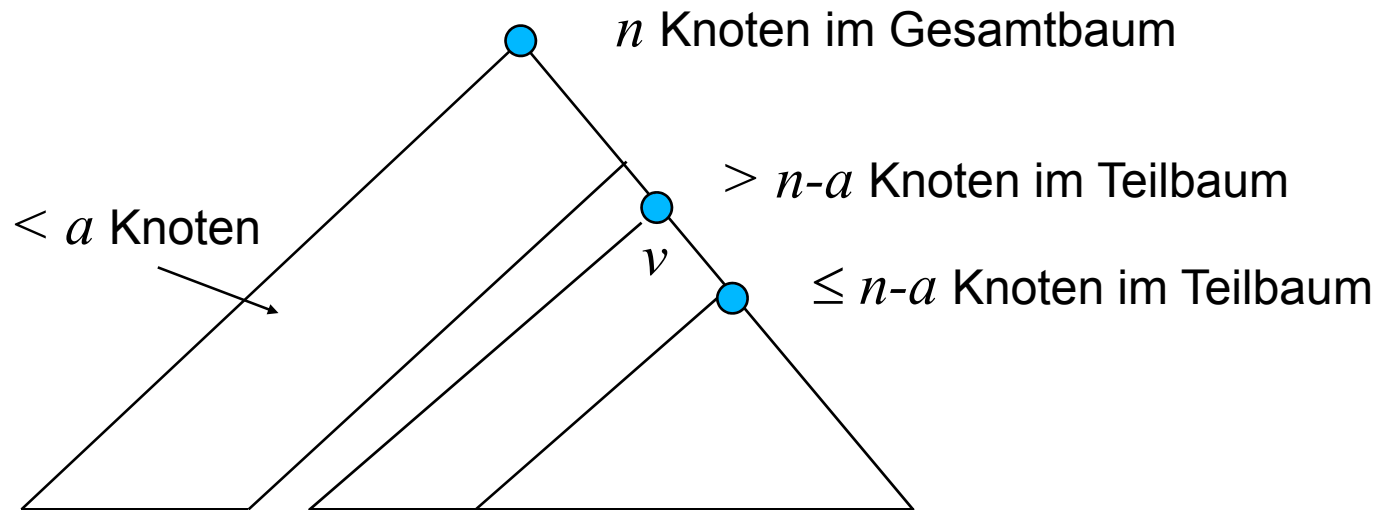
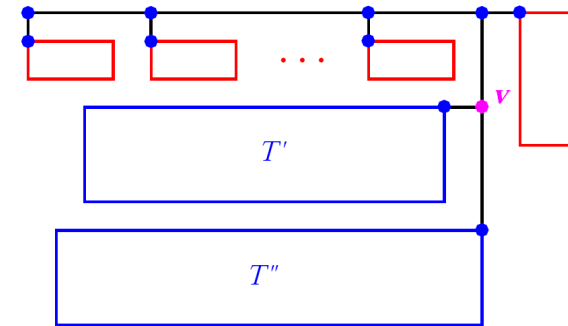


Illustration der Ungleichung

$$n(\text{rechts}(v)) \leq n - (n \log n)^{1/2} < n(v).$$

$$a = (n \log n)^{1/2}$$



$$w(n) = \max \{w(n'), w(n''), a\} + O(\log n)$$

$$h(n) = \max \{h(n') + h(n'') + O(\log n), a\}$$

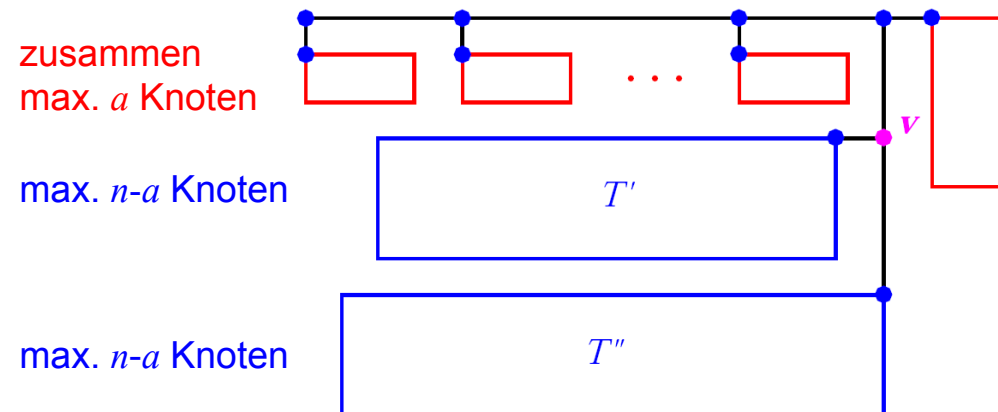
Eigenschaften des Algorithmus des rekursiven Drehens

Die Breite $w(n)$ und die Höhe $h(n)$ der Zeichnung eines Baums mit n Knoten erfüllen die Rekurrenzbeziehungen

- $w(n) = \max \{w(n'), w(n''), a\} + O(\log n)$
- $h(n) = \max \{h(n') + h(n'') + O(\log n), a\}$

wobei

- $a = (n \log n)^{1/2}$
- $\max \{n', n''\} \leq n - a$



Für die Lösungen gilt $w(n) = h(n) = O((n \log n)^{1/2})$.

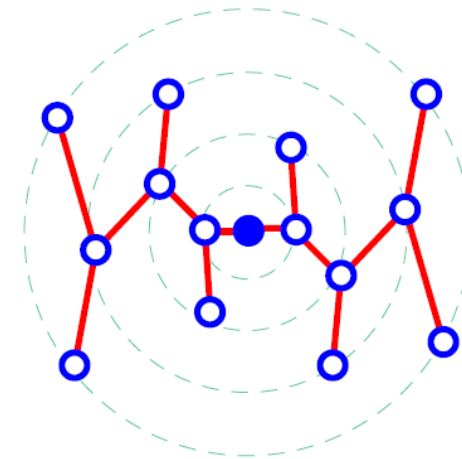
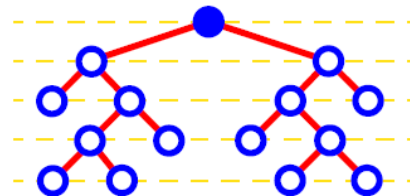
Zeichnungen von freien Bäumen

Freier Baum: ein kreisfreier zusammenhängender Graph ohne ausgezeichnete Wurzel, d.h. ohne Ausdruck einer Hierarchiebeziehung.

Beispiel: Spannbaum eines Graphen

Algorithmenskizze

- Wähle einen zentralen Knoten des Baums als Wurzel (zentraler Knoten: ein Knoten, der den kleinste maximalen Abstand zu allen Blättern hat).
- Generiere einen stufenorientierte Zeichnung, mit waagrechten oder kreisförmigen Stufen

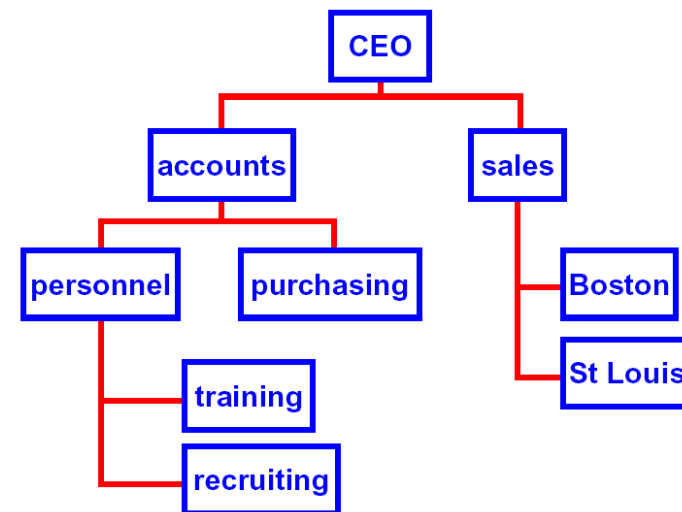


Zeichnungen mit ausgedehnten Knoten

Tip-Over-Layout

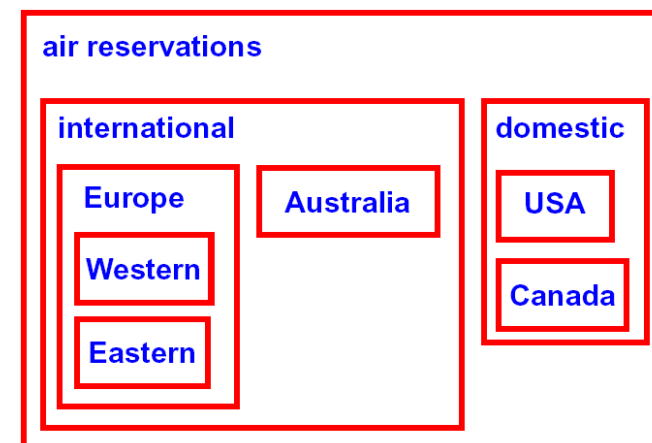
aufwärts gerichtete Darstellung, so dass die Kinder eines Knotens

- entweder waagrecht oder senkrecht angeordnet sind
- Teile der Kanten zum Elternknoten gemeinsam haben



Inklusions-Layout

stellt die Eltern-Kind-Beziehung durch Inklusion zwischen gleich ausgerichteten Rechtecken dar.



Literatur

P. Eades, T. Lin, X. Lin, Two Tree Drawing Conventions,
Journal of Computational Geometry and Appl., 3 (1993) 133-153

D.1. Graphentheorie

- D.1.1. Einleitung
- D.1.2. Gerichtete Graphen
- D.1.3. Ungerichtete Graphen
- D.1.4. Hypergraphen
- D.1.5. Speicherung
- D.1.6. Teilgraphen
- D.1.7. Pfade und Kreise
- D.1.8. Spezielle Eigenschaften

D.2. Zeichnen von Graphen

- D.2.1. Zeichenkonventionen
- D.2.2. Anforderungen
- D.2.3. Vorgehensweisen

D.3. Zeichenstrategien für Graphen

- D.3.1. Topologie-Form-Metrik-Ansatz
 - D.3.1.1. Verfeinerungsebenen
 - D.3.1.2. Algorithmus
 - D.3.1.3. Beispiel
- D.3.2. Hierarchischer Ansatz
 - D.3.2.1. Algorithmus
 - D.3.2.2. Beispiele
 - D.3.2.3. DAG
 - D.3.2.4. Bemerkungen
- D.3.3. Sichtbarkeitsansatz
 - D.3.3.1. Algorithmus
 - D.3.3.2. Beispiel
- D.3.4. Verfeinerungsansatz
 - D.3.4.1. Algorithmus
 - D.3.4.2. Beispiel

D.4. Schichtenweises Zeichnen allg. gerichteter Graphen

- D.4.1. Einleitung
- D.4.2. Algorithmus von Sugiyama
 - D.4.2.1. Übersicht
 - D.4.2.2. Entfernen von Zyklen (Schritt 1)
 - D.4.2.3. Schichtzuordnung (Schritt 2)
 - D.4.2.4. Kreuzungsreduktion (Schritt 3)
 - D.4.2.5. Waagrechte Koordinatenzuweisung (Schritt 4)
 - D.4.2.6. Bemerkungen

D.5. (Binäre) Bäume

- D.5.1. Konventionen
- D.5.2. Stufenweises Zeichnen
 - D.5.2.1. Einfacher Algorithmus
 - D.5.2.2. Algorithmus von Reingold/Tilford
- D.5.3. Horizontal-vertikal-Zeichnen
 - D.5.3.1. Algorithmus
 - D.5.3.2. Eigenschaften
- D.5.4. Zeichnen durch rekursives Drehen
 - D.5.4.1. Algorithmus
 - D.5.4.2. Eigenschaften
- D.5.5. Freie Bäume
- D.5.6. Bäume mit ausgedehnten Knoten

D.6. Vereinfachung von Polygonzügen

- D.6.1. Einführung
- D.6.2. min - # - Version (Imai, Iri)
- D.6.3. Fehlermaße
- D.6.4. min - e - Version
- D.6.5. Douglas-Peucker-Algorithmus

Vereinfachung von Polygonzügen

Gegeben: Ein Polygonzug P , $\varepsilon > 0$.

Gesucht: Ein Polygonzug P' , der dieselben Endpunkte wie P hat, nur Punkte aus P hat, von P maximal um ε abweicht und möglichst wenig Kanten hat.

Anwendung: Vereinfachung von Polygonzugbegrenzungen bei Gebieten in Karten, Internetdienste (Google-Maps), Produktionslinie, usw..



Lösungsverfahren:

Übersicht

- A. Kolesnikov, Polygonal Approximation,
http://www.cs.joensuu.fi/~koles/approximation/Ch3_0.html
- J. Peschier, Polygon Simplification – An Overview,
<http://www.jarno.demon.nl/polygon.htm>

im Folgenden: ein graphbasierter Ansatz von Imai, Iri

Literatur

H. Imai, M. Iri, Polygonal approximations of a curve – formulation and algorithms, in: Computational Morphology, Elsevier Science Publishers B.V., 1988, 71-86

**Graphbasierter Algorithmus Vereinfachung von Polygonzügen
(min - # - Version) :**

Gegeben: Ein Polygonzug $P = (\mathbf{p}_0, \dots, \mathbf{p}_{n-1})$, $\varepsilon > 0$.

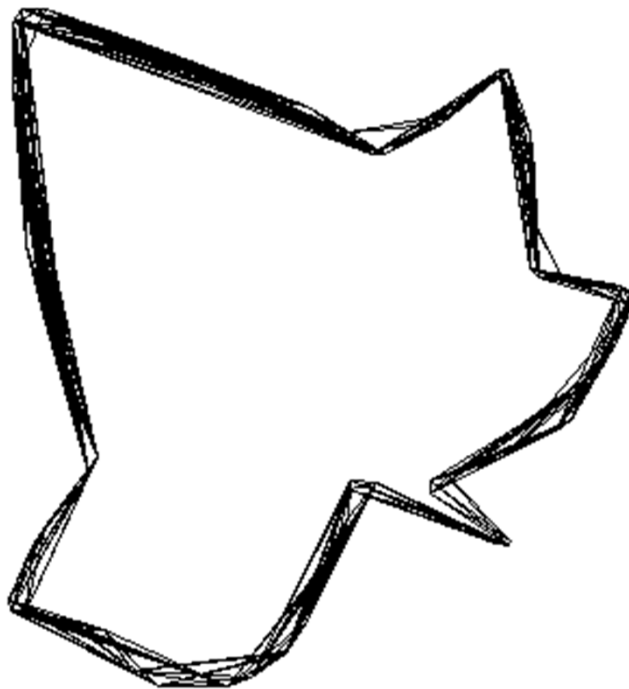
Gesucht: Ein Polygonzug P' , der dieselben Endpunkte wie P hat,
von P maximal um ε abweicht und möglichst wenig Kanten hat.

Ablauf:

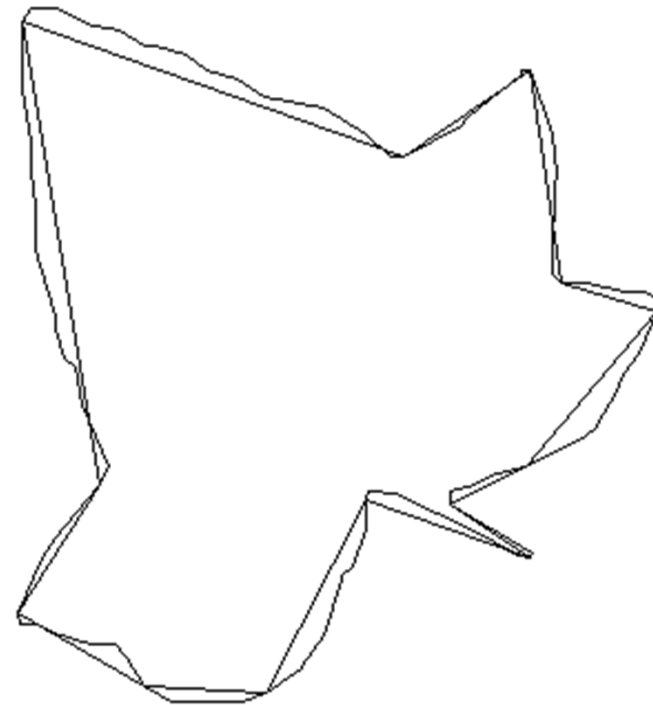
1. Konstruiere einen Graphen $G(P, \varepsilon)$, der genau dann eine gerichtete Kante von \mathbf{p}_i , \mathbf{p}_j , $i < j$, enthält, wenn die Abweichung der Strecke zwischen \mathbf{p}_i und \mathbf{p}_j von allen dazwischen liegenden Punkten \mathbf{p}_k , $k = i+1, \dots, j-1$, kleiner als ε ist.
2. Berechne einen kürzesten Weg in $G(P, \varepsilon)$ zwischen \mathbf{p}_0 sowie \mathbf{p}_{n-1} und gib diesen als Lösung zurück.

Beispiel:

Graph $G(P, \varepsilon)$ für einen gegebenen Polygonzug und gegebenes ε



Approximation mit minimaler Kantenanzahl und Toleranz ε



Bemerkungen:

1. Eine Möglichkeit der Berechnung von $G(P, \varepsilon)$ ist, für alle Punktpaare $\mathbf{p}_i, \mathbf{p}_j, i < j$, die Abweichung der dazwischen liegenden Punkte von der Kante zwischen den Punkten zu bestimmen und nur die Kanten zu behalten, deren Abweichung kleiner als ε ist.

Falls $t_a(k)$ der Zeitbedarf für den Test der Abweichung von der Kante zwischen zwei Punkten mit Indexabstand k ist, beträgt Zeitaufwand für alle Tests

$$T_a(n) = \sum_{k=1}^{n-1} (n - k) t_a(k).$$

Für $t_a(k) = O(k)$ ergibt sich der Gesamtzeitaufwand $T_a(n) = O(n^3)$.

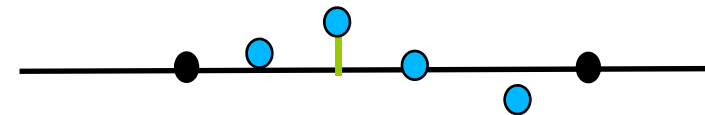
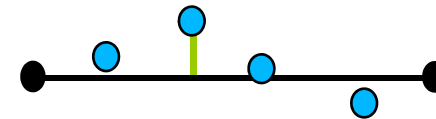
2. Ein kürzester Weg in $G(P, \varepsilon)$ lässt sich mit dem Algorithmus von Dijkstra in $O(n^2)$ Zeit berechnen.

Fehlermaße:

messen der Abweichung zwischen einer Strecke $\mathbf{p}_i, \mathbf{p}_j$ und allen dazwischen liegenden Punkten $\mathbf{p}_k, k = i + 1, \dots, j - 1$, des gegebenen Polygonzugs P .

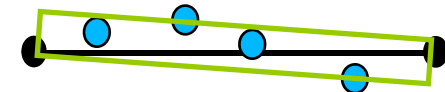
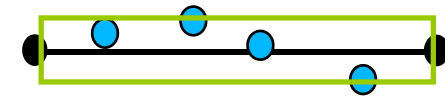
Beispiele:

- **Punkt-Streckenabstand:**
der maximale Abstand zwischen der Strecke und allen dazwischenliegenden Punkten
- **Punkt-Geradenabstand** („Parallelstreifenkriterium“):
der maximale Abstand zwischen der Geraden und allen dazwischenliegenden Punkten



Bemerkung: der maximale Abstand kann auch durch ein anderes Kriterium, z.B. die mittlere Quadratsumme aller Abstände, ersetzt werden.

- **streckenausgerichtetes Hüllrechteck:**
die Hälfte der Länge der zur Strecke senkrechten Kanten des Rechtecks, das längs der Strecke ausgerichtet ist und dessen Rand \mathbf{p}_i und \mathbf{p}_j enthält
- **minimales Hüllrechteck:**
die Hälfte der Länge der kürzesten Kante des Rechtecks mit kleinster minimaler Kantenlänge, das die gegebenen Punkte \mathbf{p}_k , $k = i, \dots, j$, enthält.



Bemerkung:

Für die genannten Kriterien gibt es Algorithmen, die den Graphen $G(P, \varepsilon)$ in $O(n^2 \log n)$ Zeit berechnen.

Literatur:

- H. Imai, M. Iri, Polygonal approximations of a curve – formulation and algorithms, in: Computational Morphology, Elsevier Science Publishers B.V., 1988, 71-86
- A. Melkman, J. O'Rourke, On polygonal chain approximation, in: Computational Morphology, Elsevier Science Publishers B.V., 1988, 87-95
- D. Eu, G.T. Toussaint, On approximating polynomial curves in two and three dimensions, Computer Vision, Graphics and Image Processing (CVGIP): Graphical Models and Image Processing 56(3), 1994, 231-246

Bemerkung:

Eine Version des Problems zur Vereinfachung von Polygonzügen ist, bei vorgegebener Kantenanzahl einen Polygonzug mit geringster Abweichung bezüglich eines gegebenen Fehlermaßes zu finden:

Vereinfachung von Polygonzügen (min – e - Version)

Gegeben: Ein Polygonzug P , $m > 0$, ein Fehlermaß.

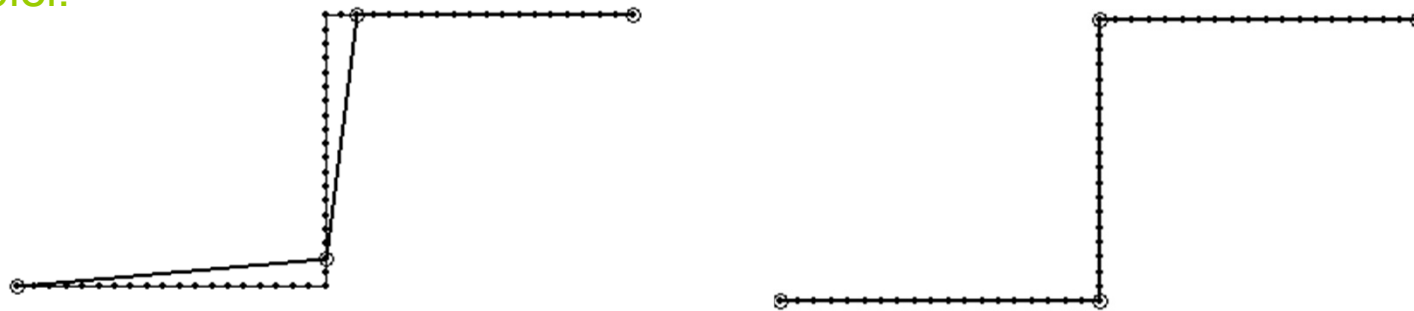
Gesucht: Ein Polygonzug P' , der dieselben Endpunkte wie P hat, m Punkte enthält, die aus P stammen und möglichst wenig von P bezüglich des Fehlermaßes abweicht.

Ablauf:

1. Konstruiere einen gewichteten gerichteten Graphen $G(P)$, der alle gerichtete Kanten von $\mathbf{p}_i, \mathbf{p}_j, i < j$, enthält, welche die maximale Abweichung der dazwischen liegenden Punkte $\mathbf{p}_k, k = i + 1, \dots, j - 1$, als Gewicht haben.
2. Finde durch binäre Suche über alle auftretenden Gewichte ein minimales Gewicht e , sodass ein kürzester Weg zwischen \mathbf{p}_0 und \mathbf{p}_{n-1} im Untergraphen $G(P, e)$ existiert, der höchstens m Knoten hat. Gib einen solchen Weg zurück, falls einer existiert.

Bemerkung

Durch min - e- Version können Lösungen der min - # - Version verbessert werden.

Beispiel:

Zeitaufwand

Für die meisten der genannten Fehlerkriterien gibt es Algorithmen, die die min - e – Version mit $O(n^2 \log n)$ Zeitaufwand lösen.

Literatur:

siehe Angaben zu Algorithmen für die min - # - Version

Bemerkung:

Ein weit verbreitetes Verfahren zur Vereinfachung ist der **Douglas-Peucker-Algorithmus**, bei welchem es sich um ein Top-Down Divide-and-Conquer-Verfahren handelt.

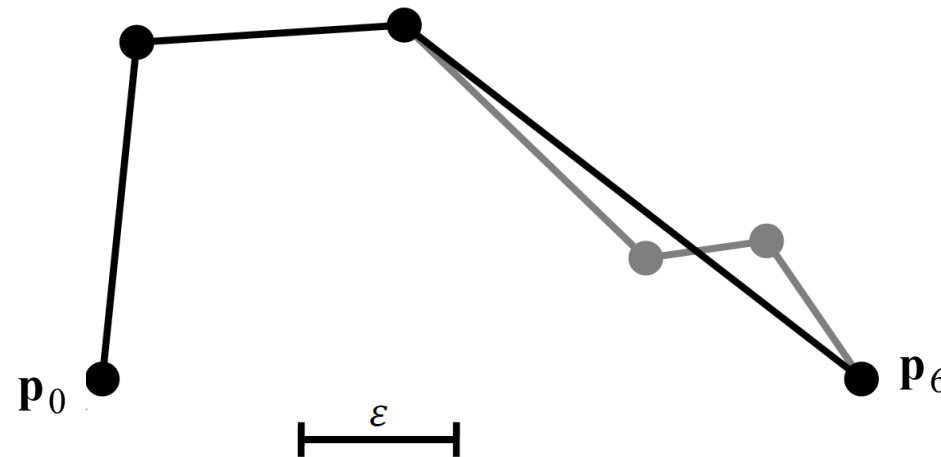
Literatur:

David Douglas, Thomas Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, The Canadian Cartographer 10, 1973, 112-122.

Douglas-Peucker-Algorithmus: Divide-and-Conquer-Verfahren

Gegeben: Ein Polygonzug $P = (p_0, \dots, p_{n-1})$, $\varepsilon > 0$.

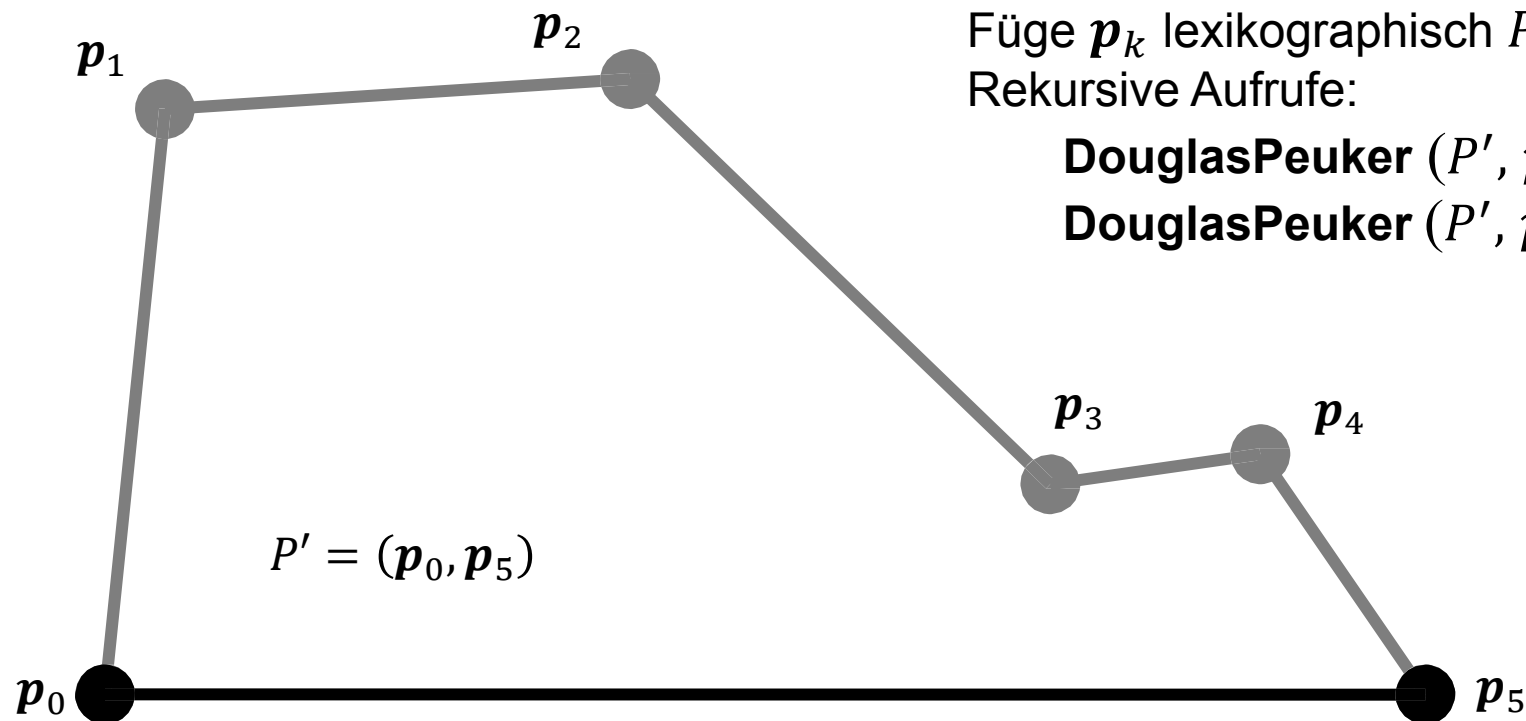
Gesucht: Ein Polygonzug P' , der dieselben Endpunkte wie P hat, von P maximal um ε abweicht

**Literatur:**

David Douglas, Thomas Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, The Canadian Cartographer 10, 1973, 112-122.

Algorithmus $P' = (p_0, p_{n-1})$ **DouglasPeucker** (P', p_0, p_{n-1})

Beispiel:

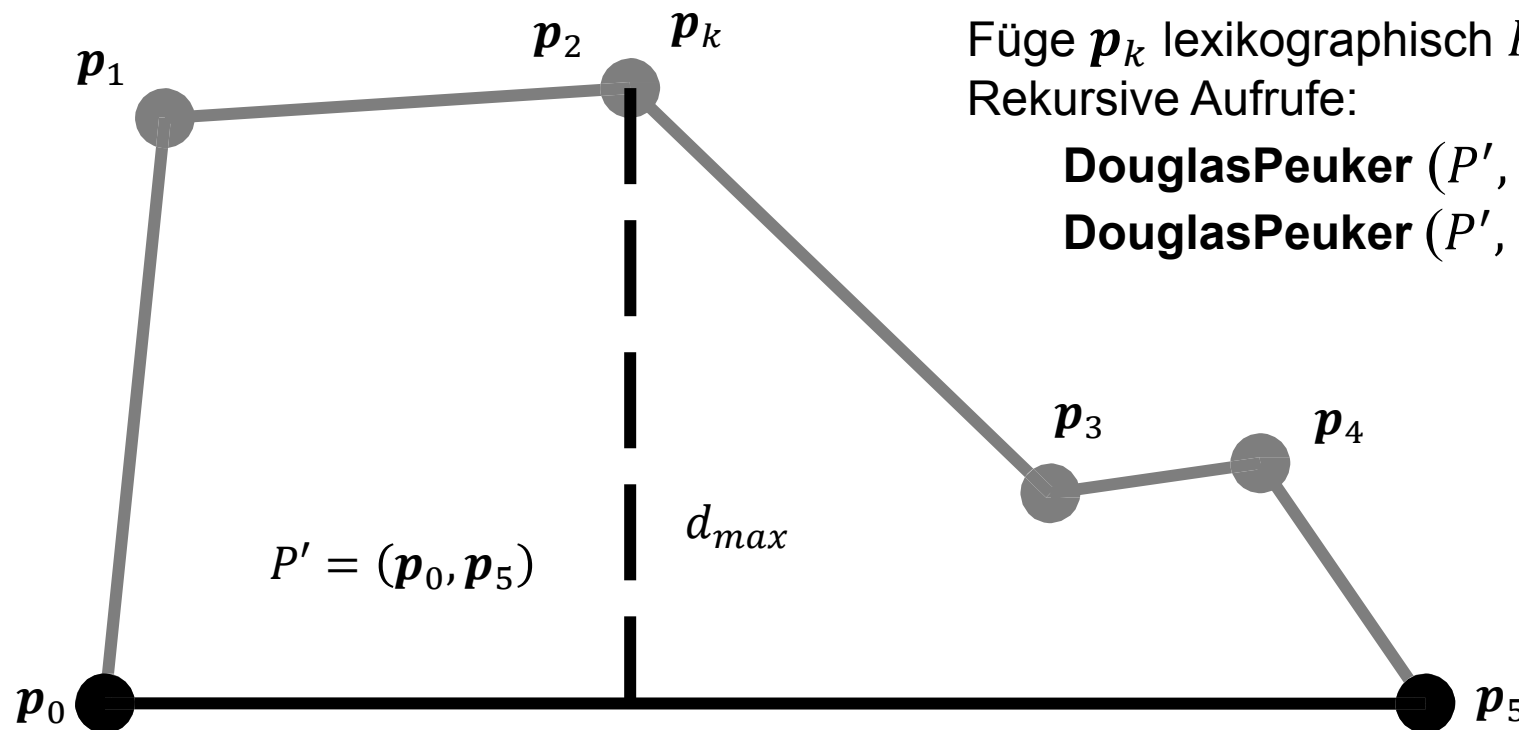
 $P = (p_0, p_1, p_2, p_3, p_4, p_5), \varepsilon$ **DouglasPeucker** (P', p_i, p_j) $d_{max} = -\infty, K = 0$ Für $k = i + 1, \dots, j - 1$ $d = \text{Abstand } p_k \text{ zu Segment } (p_i, p_j)$ Falls $d > d_{max}$ dann $d_{max} = d, K = k$ Falls $d > \varepsilon$ dannFüge p_k lexikographisch P' hinzu

Rekursive Aufrufe:

DouglasPeucker (P', p_i, p_K)**DouglasPeucker** (P', p_K, p_j)

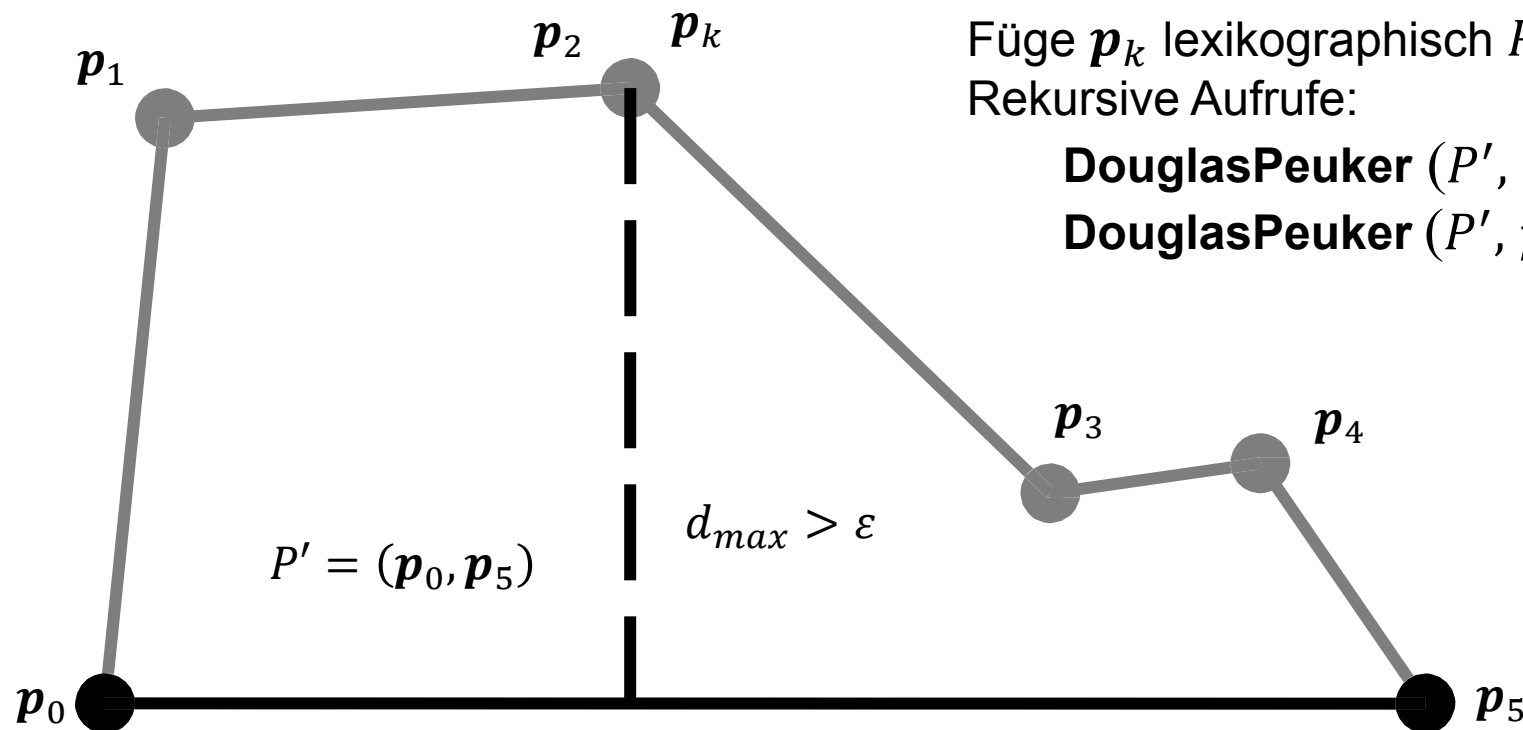
Algorithmus $P' = (p_0, p_{n-1})$ **DouglasPeuker** (P', p_0, p_{n-1})**DouglasPeuker** (P', p_i, p_j) $d_{max} = -\infty, K = 0$ Für $k = i + 1, \dots, j - 1$ $d = \text{Abstand } p_k \text{ zu Segment } (p_i, p_j)$ Falls $d > d_{max}$ dann $d_{max} = d, K = k$ Falls $d > \varepsilon$ dannFüge p_k lexikographisch P' hinzu

Rekursive Aufrufe:

DouglasPeuker (P', p_i, p_K)**DouglasPeuker** (P', p_K, p_j)

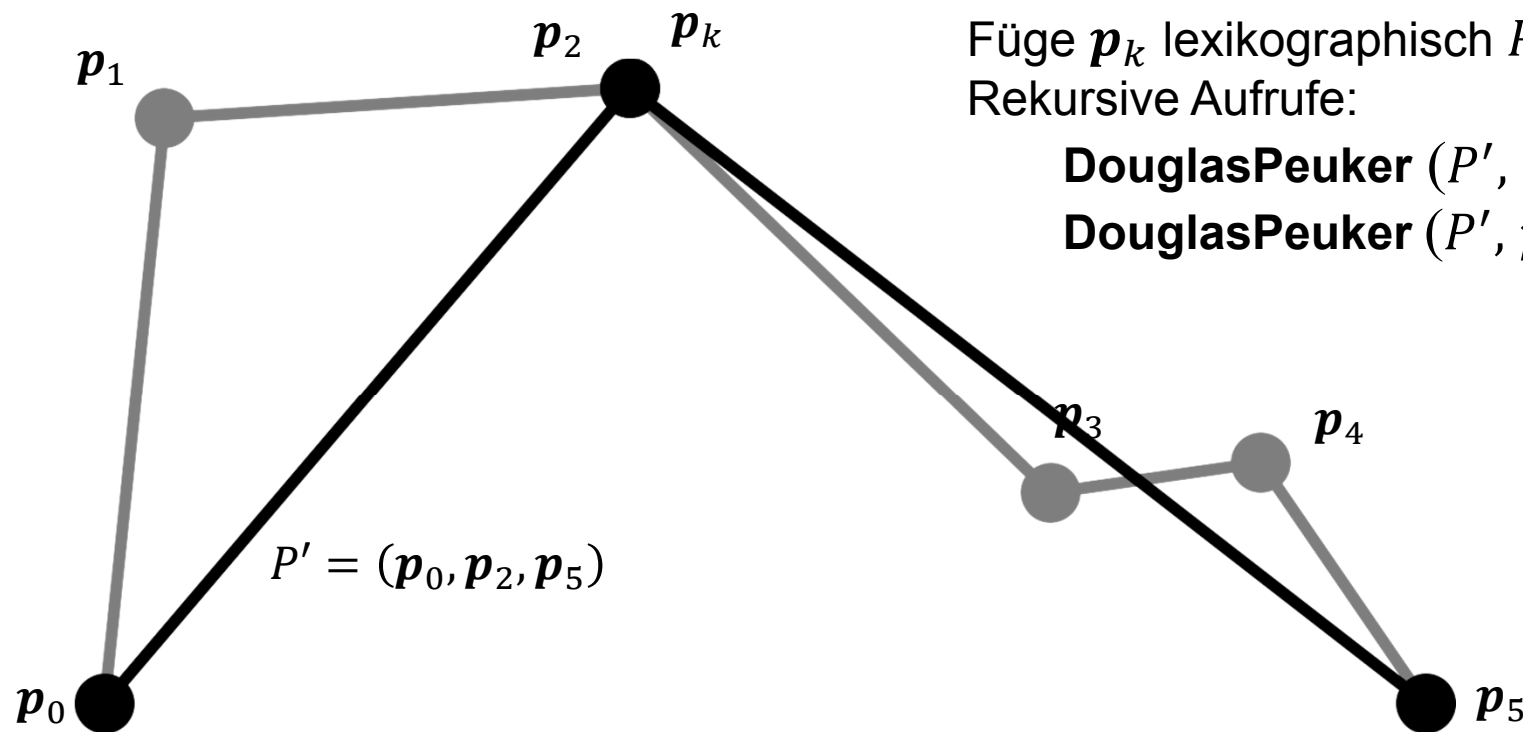
Algorithmus $P' = (p_0, p_{n-1})$ **DouglasPeuker** (P', p_0, p_{n-1})**DouglasPeuker** (P', p_i, p_j) $d_{max} = -\infty, K = 0$ Für $k = i + 1, \dots, j - 1$ $d = \text{Abstand } p_k \text{ zu Segment } (p_i, p_j)$ Falls $d > d_{max}$ dann $d_{max} = d, K = k$ Falls $d > \varepsilon$ dannFüge p_k lexikographisch P' hinzu

Rekursive Aufrufe:

DouglasPeuker (P', p_i, p_K)**DouglasPeuker** (P', p_K, p_j)

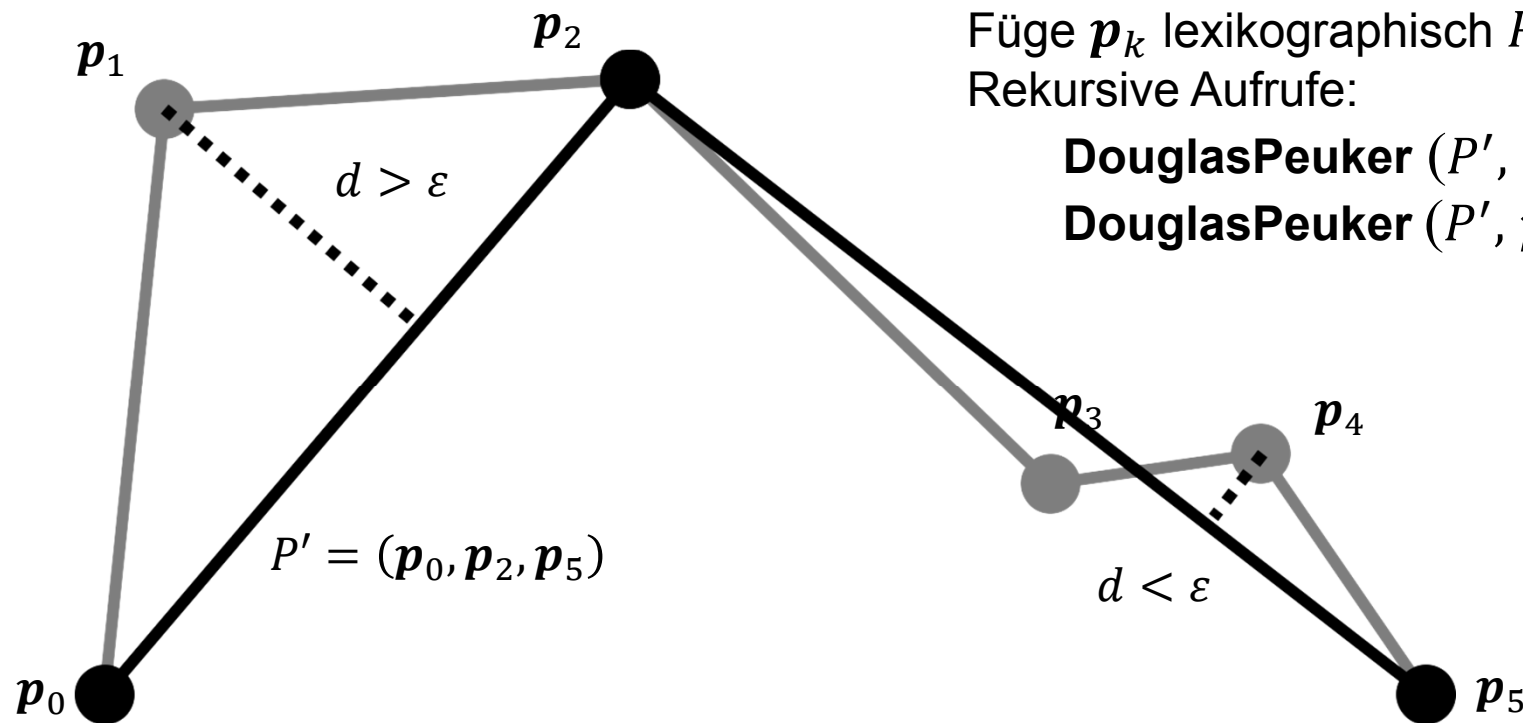
Algorithmus $P' = (p_0, p_{n-1})$ **DouglasPeucker** (P', p_0, p_{n-1})**DouglasPeucker** (P', p_i, p_j) $d_{max} = -\infty, K = 0$ Für $k = i + 1, \dots, j - 1$ $d = \text{Abstand } p_k \text{ zu Segment } (p_i, p_j)$ Falls $d > d_{max}$ dann $d_{max} = d, K = k$ Falls $d > \varepsilon$ dannFüge p_k lexikographisch P' hinzu

Rekursive Aufrufe:

DouglasPeucker (P', p_i, p_K)**DouglasPeucker** (P', p_K, p_j)

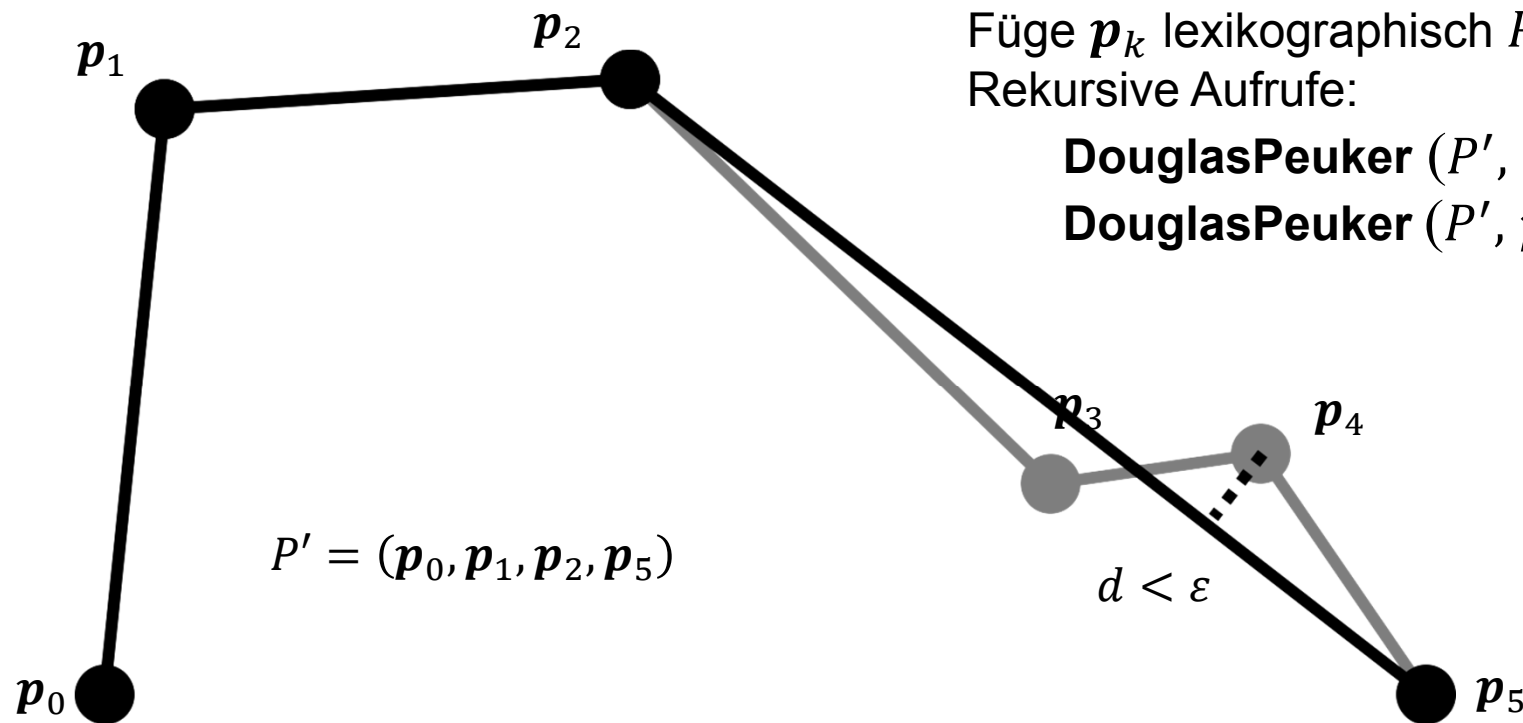
Algorithmus $P' = (p_0, p_{n-1})$ **DouglasPeuker** (P', p_0, p_{n-1})**DouglasPeuker** (P', p_i, p_j) $d_{max} = -\infty, K = 0$ Für $k = i + 1, \dots, j - 1$ $d = \text{Abstand } p_k \text{ zu Segment } (p_i, p_j)$ Falls $d > d_{max}$ dann $d_{max} = d, K = k$ Falls $d > \varepsilon$ dannFüge p_k lexikographisch P' hinzu

Rekursive Aufrufe:

DouglasPeuker (P', p_i, p_K)**DouglasPeuker** (P', p_K, p_j)

Algorithmus $P' = (p_0, p_{n-1})$ **DouglasPeuker** (P', p_0, p_{n-1})**DouglasPeuker** (P', p_i, p_j) $d_{max} = -\infty, K = 0$ Für $k = i + 1, \dots, j - 1$ $d = \text{Abstand } p_k \text{ zu Segment } (p_i, p_j)$ Falls $d > d_{max}$ dann $d_{max} = d, K = k$ Falls $d > \varepsilon$ dannFüge p_k lexikographisch P' hinzu

Rekursive Aufrufe:

DouglasPeuker (P', p_i, p_K)**DouglasPeuker** (P', p_K, p_j)

Algorithmus $P' = (p_0, p_{n-1})$ **DouglasPeuker** (P', p_0, p_{n-1})**DouglasPeuker** (P', p_i, p_j) $d_{max} = -\infty, K = 0$ Für $k = i + 1, \dots, j - 1$ $d = \text{Abstand } p_k \text{ zu Segment } (p_i, p_j)$ Falls $d > d_{max}$ dann $d_{max} = d, K = k$ Falls $d > \varepsilon$ dannFüge p_k lexikographisch P' hinzu

Rekursive Aufrufe:

DouglasPeuker (P', p_i, p_K)**DouglasPeuker** (P', p_K, p_j)