Charakterisierung Baum

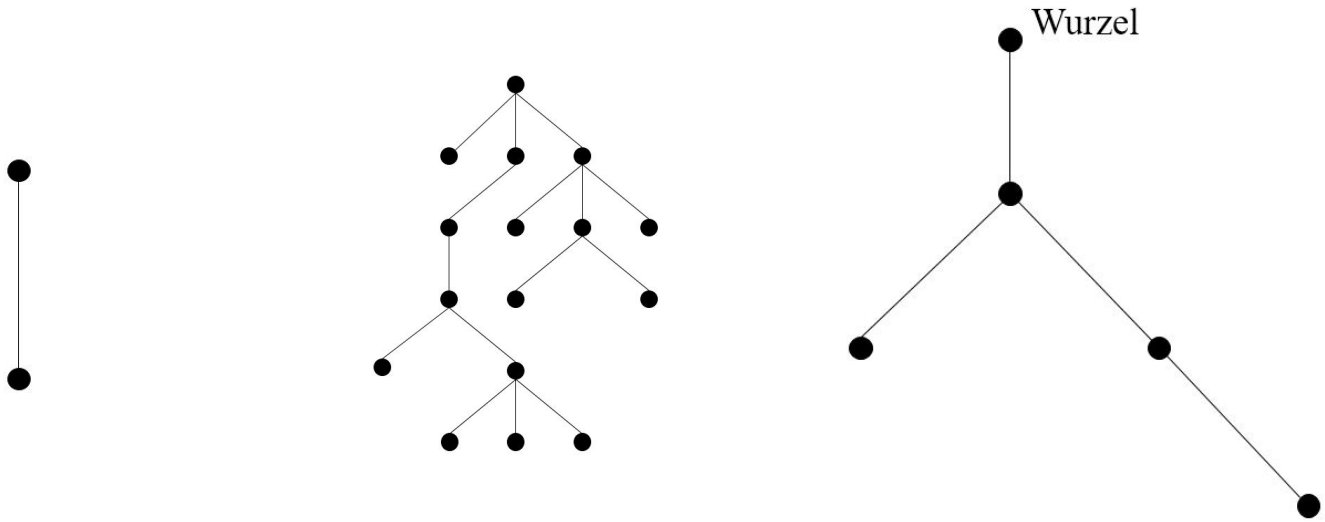
Für einen Graphen G mit n Knoten und m Kanten sind die folgenden Aussagen äquivalent:

1. G ist ein Baum
2. G ist ein zusammenhängender Graph ohne Kreise
3. G ist zusammenhängend, aber entfernt man eine beliebige Kante, so zerfällt G in zwei Zusammenhangskomponenten
4. G ist zusammenhängend und $n = m + 1$
   (G hat einen Knoten mehr als Kanten)

Wurzelbaum

Ein Wurzelbaum ist ein Baum, in dem ein Knoten ausgezeichnet wird: die *Wurzel*. Jeder Knoten eines Baumes kann Wurzel werden.

Die Baumwurzel wird i.d.R. oben gezeichnet, alle Kanten weisen nach unten.



Wurzelbaum – Definitionen

- Sind x und y duch eine Kante verbunden und x ist näher zur Wurzel als y, so ist x **Vater** von y und y **Sohn** von x.
- Existiert ein Weg von y zur Wurzel, welcher x enthält, so heißt x **Vorfahre** von y und y **Nachkomme** von x.
  - Der einzige Knoten ohne Vorfahren ist die Wurzel!
- Knoten mit Nachkommen heißen **innere Knoten**.
- Knoten ohne Nachkommen heißen **Blätter** (= die von der Wurzel verschiedenen Knoten mit Grad 1).
- Ist die *Reihenfolge* bei den Söhnen von Bedeutung, so spricht man von einem **geordneten Wurzelbaum**. Die Söhne werden (meist von links nach rechts) durchnummeriert: erster Sohn, zweiter Sohn, usw.

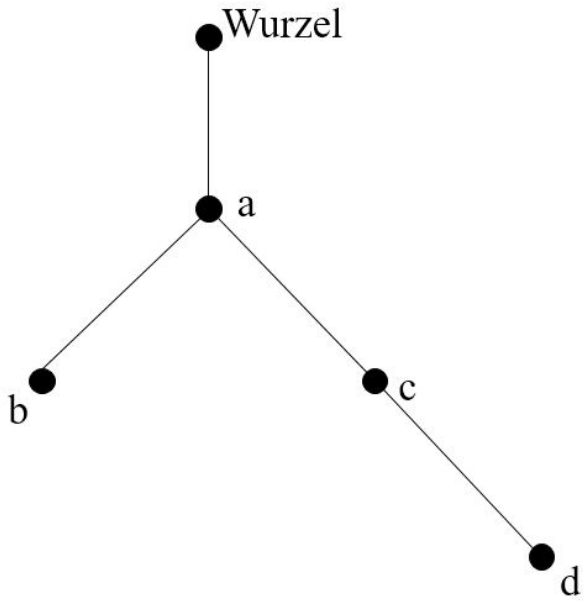
Wurzelbaum

Beachten Sie:

In einem Wurzelbaum bildet jeder Knoten mit allen seinen Nachkommen und den dazugehörigen Kanten wieder einen Wurzelbaum.

$\rightarrow$ Zusammenhang zur Rekursion!

Beispiel: Wurzelbaum



- *a* ist Vater von *b*
- *d* ist Nachkomme von *a*
- *d* ist kein Nachkomme von *b*
- *b* und *d* sind Blätter
- *a* und *c* sind innere Knoten

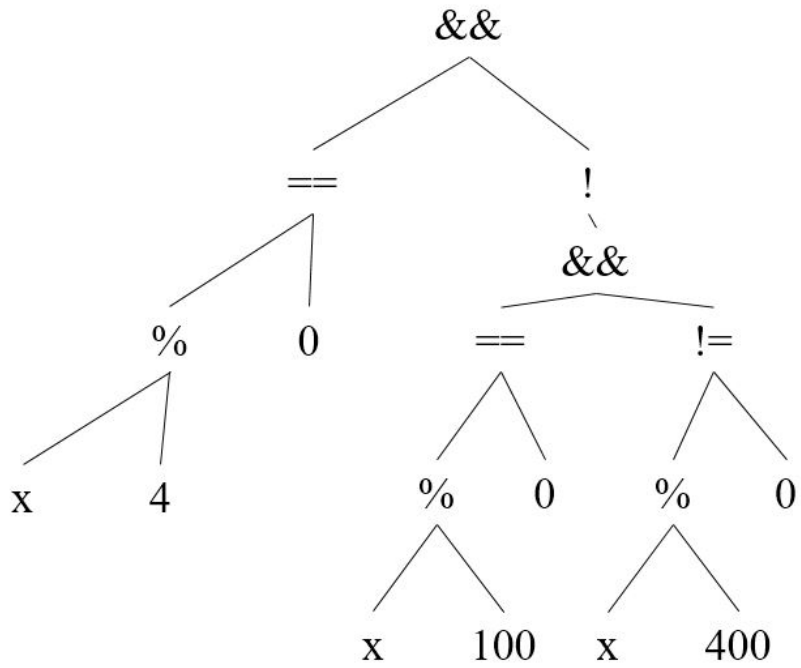
Anwendung: Compiler

Syntaxanalyse: Der Compiler überträgt den zu verarbeitenden Code eines Programms in einen Syntaxbaum (= geordneter Wurzelbaum):

- Die inneren Knoten stellen die Operatoren dar;
- die Operanden werden aus den Teilbäumen gebildet, die an den Söhnen dieses Operators hängen.

Syntaxbaum für

$(x\%4 == 0)\ \&\&$
$!((x\%100 == 0)\ \&\&\ (x\%400\ != 0))$



Anwendung: Compiler

Nach dem Aufbau des Syntaxbaums kann die Auswertung des Ausdrucks *rekursiv* erfolgen:

- Um den Baum auszuwerten, müssen alle Teilbäume in den Söhnen der Wurzel ausgewertet werden.
- Anschliessend kann die Operation der Wurzel durchgeführt werden.
- Rekursion endet, wenn die Wurzel selbst ein Operand ist.

Wurzelbäume

- Das **Niveau** eines Knotens in einem Wurzelbaum ist die Anzahl der Knoten des Weges von der Wurzel ist zu diesem Knoten
  - Die Wurzel selbst hat das Niveau 1.
- Das maximale Niveau aller Knoten heißt **Höhe** des Wurzelbaumes.
- Hat in einem Wurzelbaum jeder Knoten maximal n Söhne, so heißt er **n-ärer Wurzelbaum**
- Hat jeder Knoten *genau* n oder 0 Söhne, so heißt er **regulärer n-ärer Wurzelbaum**
  - $n = 2 \rightarrow$ **binärer (regulärer) Wurzelbaum**, die Söhne werden *linke* und *rechte* Söhne genannt.
- In einem binären Wurzelbaum der Höhe H mit n Knoten gilt:

$$H \geq log_2(n+1)$$

Suchbäume

Vorige Aussage impliziert:

- In einem binären Wurzelbaum der Höhe H lassen sich bis zu $2^H - 1$ Knoten unterbringen (z.B. $H = 18 \rightarrow$ ca. 262 000 Knoten!)
- Zu jedem dieser Knoten gibt es von der Wurzel aus einen Weg, der höchstens 18 Knoten besucht.

$\Rightarrow$ Datenspeicherung in den Knoten von Bäumen, auf die sehr schnell zugegriffen werden kann

$\Rightarrow$ Suchbäume

Suchbäume

- Jedem Datensatz wird ein (eindeutiger) Schlüssel zugeordnet.
- *Eintragen eines Schlüssels in den Suchbaum*: alle Schlüssel des linken Teilbaums des Knotens p sind kleiner als der Schlüssel von p, die des rechten Teilbaums sind größer als der von p.

Eintragen eines Schlüssels s

Existiert der Baum noch nicht $\Rightarrow$ erzeuge die Wurzel und trage s bei der Wurzel ein.

Ansonsten: Sei w der Schlüssel der Wurzel.

$s < w \Rightarrow$ Trage Schlüssel s im linken Teilbaum der Wurzel ein.

$s > w \Rightarrow$ Trage Schlüssel s im rechten Teilbaum der Wurzel ein.

Suche nach einem Schlüssel s

Die Suche beginnt bei Knoten x mit dem Schlüssel s_x.

$s = s_x \Rightarrow$ Schlüssel gefunden.

$s < s_x \Rightarrow$ Setze die Suche beim linken Sohn von x fort.

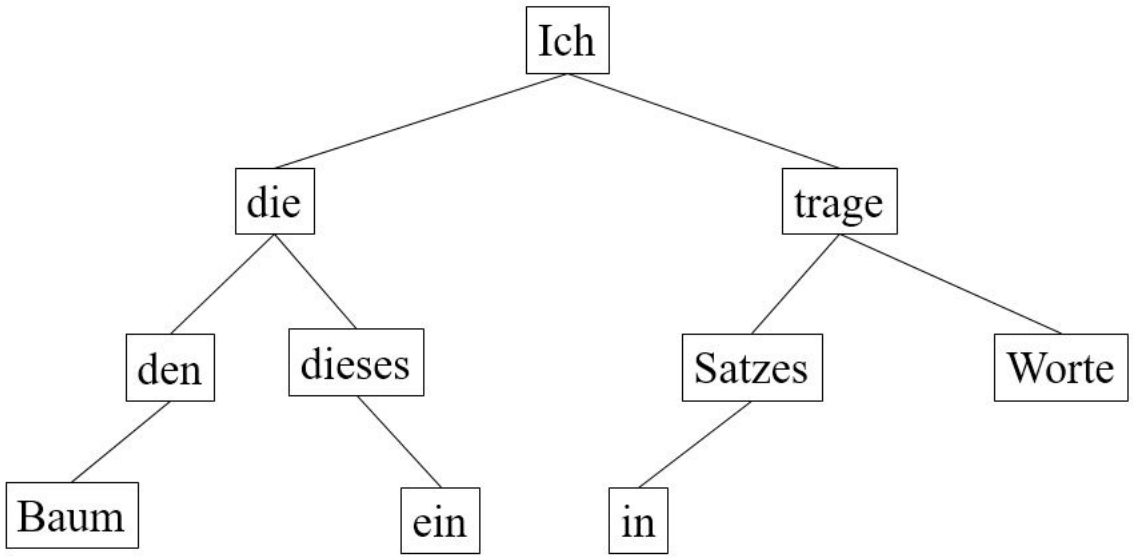
$s > s_x \Rightarrow$ Setze die Suche beim rechten Sohn von x fort.

Erweiterung für den Fall, dass s nicht vorhanden ist.

Beispiel: Suchbäume

Suchbaum für „Ich trage die Worte dieses Satzes in den Baum ein“:

- Worte werden den Knoten zugeordnet.
- Schlüssel sind die Worte selber, alphabetische Ordnung.



Anmerkung: Suche ist nur dann effizient, wenn sich die Höhen linker und rechter Teilbäume möglichst wenig unterscheiden $\Rightarrow$ *höhenbalancierte Bäume*

Traversierungsverfahren – Durchlaufen von Suchbäumen

Inorder-Durchlauf: Die Daten des Baums im Beispiel mit Wurzel x werden in alphabetischer Reihenfolge ausgegeben.

1. Wenn es einen linken Sohn gibt: Gib den Baum mit Wurzel linker Sohn von x alphabetisch aus.
2. Gib den Datensatz des Knotens x aus.
3. Wenn es einen rechten Sohn gibt: Gib den Baum mit Wurzel rechter Sohn von x alphabetisch aus.

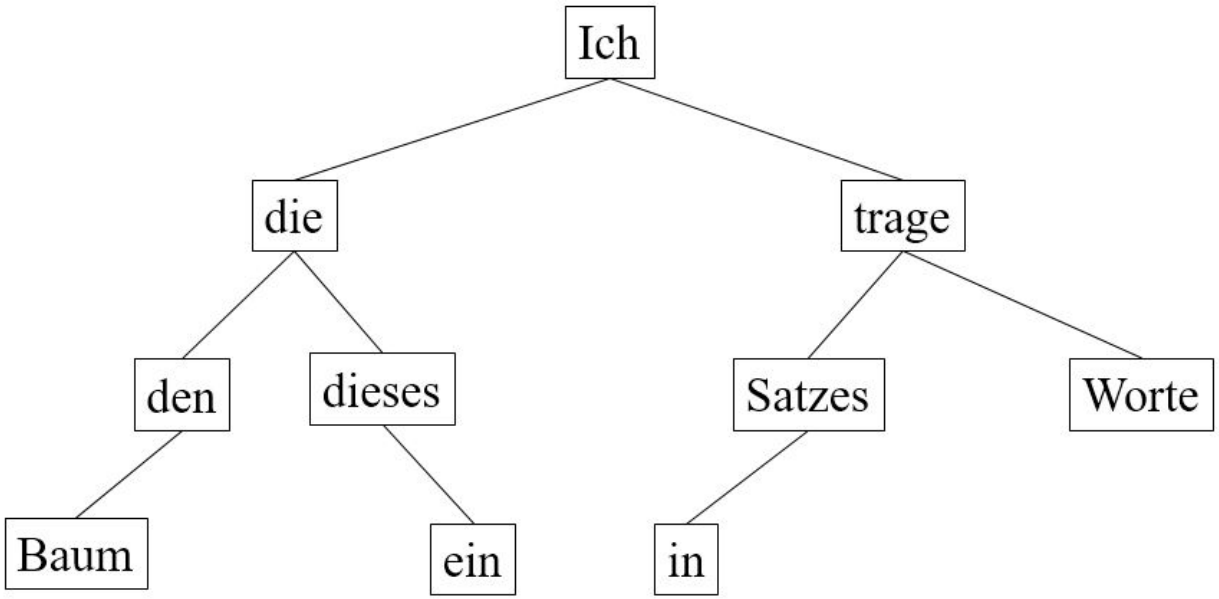
Ergebnis im Beispiel: „Baum den die dieses ein Ich in Satzes trage Worte“

Traversierungsverfahren – Durchlaufen von Suchbäumen

Inorder (L-W-R): „Baum den die dieses ein Ich in Satzes trage Worte“

Preorder (W-L-R): „Ich die den Baum dieses ein trage Satzes in Worte“

Postorder (L-R-W): „Baum den dieses ein die in Satzes Worte trage Ich“



Anwendung binäre Bäume: Huffman-Code

Grundidee: Kapazität bei Datenübertragung einsparen, indem häufig vorkommende Zeichen durch kurze Codeworte, selten vorkommende Zeichen durch längere Codeworte ersetzt werden (vgl. Morsealphabet!)

Ziel: Minimierung von benötigtem Speicherplatz durch *variable* Codewortlänge (nicht ASCII!), häufige Zeichen möglichst kurz codieren

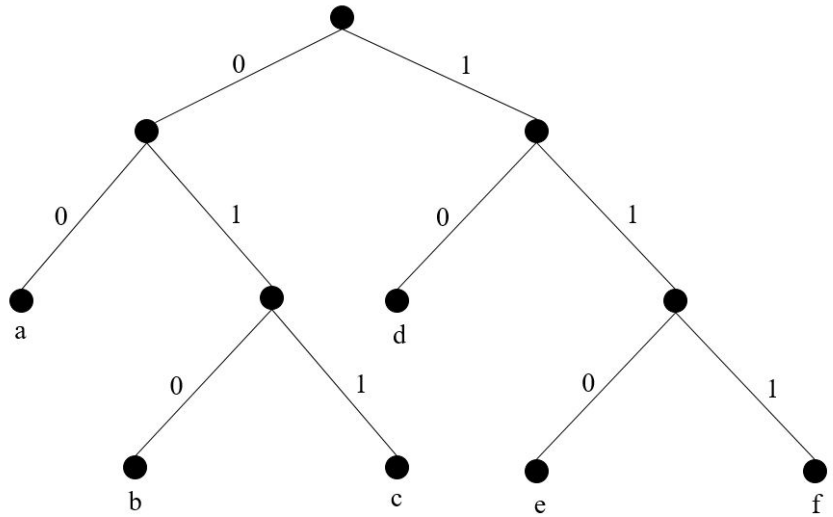
Problem: Trennung von Codewörtern („wo hört eines auf, wo beginnt das nächste“)

Lösung: *Präfix-Codes* – kein Codewort ist Anfangsteil eines anderen

Präfix Codes

Konstruktion von binären Präfix-Codes mittels Wurzelbäumen

- Codeworte = Kantenfolgen, bestehen nur aus „0“ und „1“
- Blätter: Zeichen, die kodiert werden sollen



$\Rightarrow$ Kein Codewort ist Anfangsteil eines anderen (nur Blätter kodiert!)

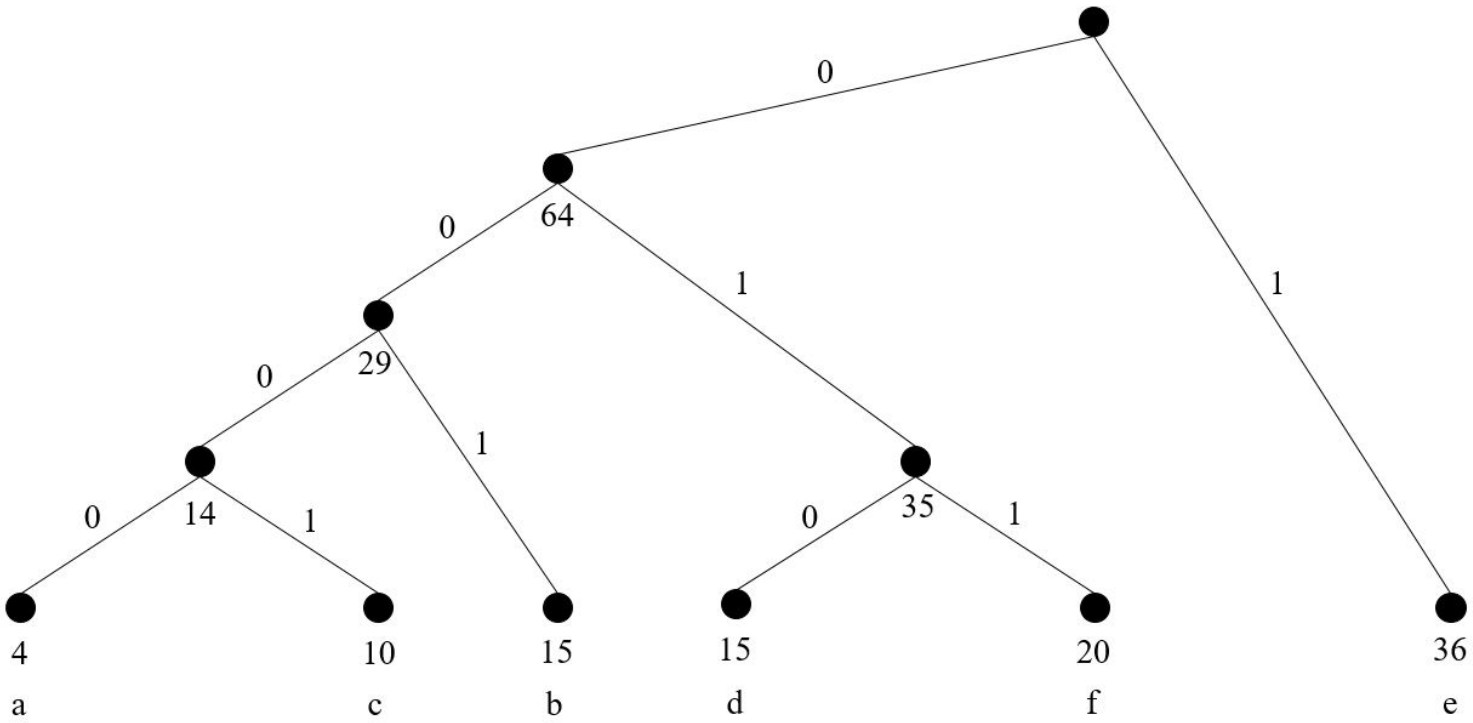
Bsp.: „0100110010“ eindeutig auflösbar in „010 011 00 10“

Anwendung binäre Bäume: Huffman-Code

Häufig vorkommende Zeichen kurz codieren, seltene Zeichen längere Codes $\Rightarrow$ Huffman-Algorithmus (produziert optimalen Präfix-Code)

1. Gegeben: Quellalphabet mit Häufigkeit der Zeichen, z. B.
   $a : 4\%$ $b : 15\%$ $c : 10\%$ $d : 15\%$ $e : 36\%$ $f : 20\%$
2. Die Zeichen werden die Blätter eines binären Baumes. Den Blättern ordnen wir die Häufigkeiten der Zeichen zu.
3. Suche die zwei kleinsten Knoten ohne Vater. Falls es mehrere gibt, wähle zwei beliebige. Erstelle einen neuen Vaterknoten zu diesen beiden Knoten und ordne diesem die Summe deren Häufigkeiten zu.
4. Wiederhole Schritt 3 so oft, bis nur noch ein Knoten ohne Vater übrig ist. Dieser ist die Wurzel des Baums.

Beispiel: Huffman-Code



Anwendung binäre Bäume: Huffman-Code

- Der Huffman-Code wird auch bei modernen Kompressionsverfahren eingesetzt (meist als Teil mehrstufiger Algorithmen).
- z. B. bei der Faxcodierung oder als Teil des jpeg-Algorithmus zur Bildkomprimierung

Durchlaufen von Graphen

Durchlaufen von Graphen

Es gibt verschiedene Zielsetzungen, wie man einen Graphen durchlaufen möchte, z. B.:

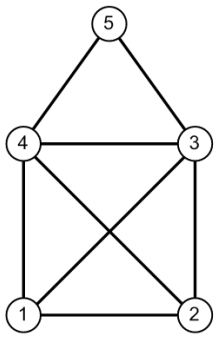
- Jeden Knoten eines Graphen genau einmal besuchen
- Jede Kante eines Graphen genau einmal besuchen
- Am Ende wieder beim Ausgangspunkt landen
- Den kürzesten Weg von A nach B finden

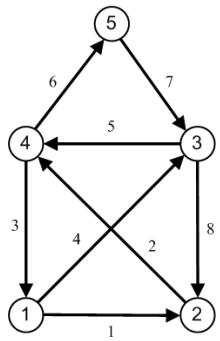
Eulersche Linie

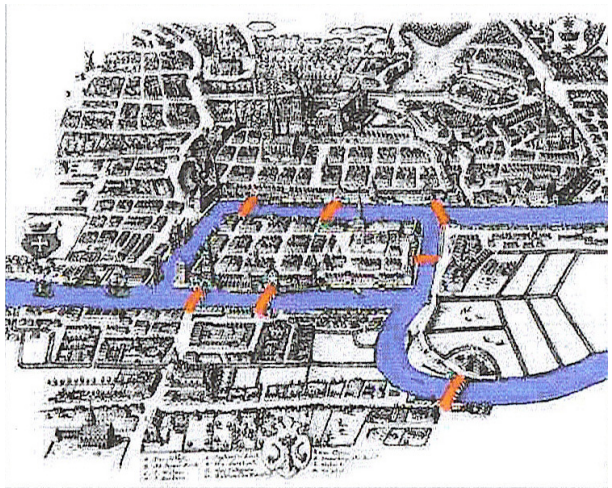
Definition

Eine Kantenfolge in einem Graphen G heißt Eulersche Linie, wenn sie jede Kante von G genau einmal enthält.

- „Haus des Nikolaus“
- Königsberger Brückenproblem: gibt es einen Weg, bei dem man alle sieben Brücken genau einmal überquert, und wenn ja, ist auch ein Rundweg möglich?







Eulerkreis

Definition

Eine Eulersche Linie heißt dann **geschlossen** oder ein Eulerkreis, wenn der Endknoten gleich dem Anfangsknoten ist.

Man kann zeigen:

- Ein (ungerichteter) Graph G besitzt genau dann eine Eulersche Linie, wenn G zusammenhängend ist und entweder alle Knotengrade gerade oder genau zwei ungerade sind.
- Ein (ungerichteter) Graph G besitzt genau dann eine geschlossene Eulersche Linie, wenn G zusammenhängend ist und alle Knotengrade gerade sind.

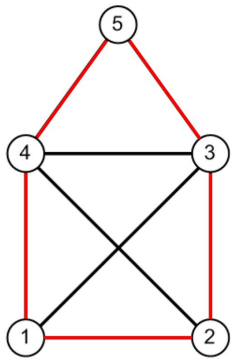
Hamiltonsche Linie

Definition

Eine Kantenfolge in einem Graphen G heißt Hamiltonsche Linie, wenn sie jeden Knoten von G genau einmal enthält. Eine Hamiltonsche Linie heißt dann **geschlossen** (Hamiltonkreis), wenn der Endknoten gleich dem Anfangsknoten ist.

Problem des Handlungsreisenden (traveling salesman problem): Reihenfolge für den Besuch mehrerer Orte so wählen, dass die gesamte Reisestrecke möglichst kurz und erste Station gleich letzte Station ist.

- Sehr schwieriges Problem (NP-vollständig)



Tiefensuche

Zurück zum Problem, alle Knoten eines beliebigen Graphen G genau einmal zu besuchen.
Tiefensuche – rekursiv, beginne bei beliebigem Knoten x von G:

Durchlaufe den Graphen ab dem Knoten x:

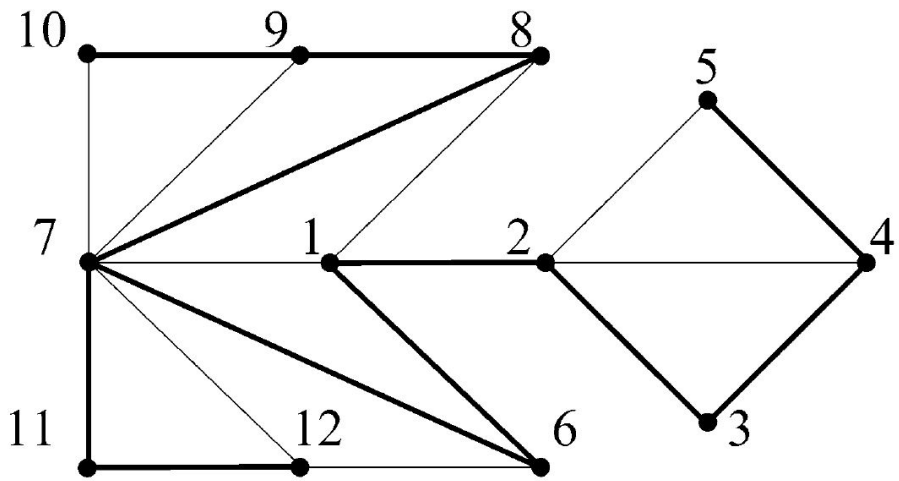
1. Markiere x als besucht
2. Für alle noch nicht besuchten Knoten y, die zu x adjazent: gehe zu y
3. Durchlaufe den Graphen ab dem Knoten y

Ausgehend von x folgt man einem Weg, bis zum ersten Knoten ohne unbesuchte Nachbarn. Dann kehrt man um bis zum nächsten Knoten mit unbesuchten Nachbarn. Einen solchen besucht man als nächstes und geht von dort aus wieder, so weit man kommt, dann wieder zurück, und so fort.

Bei binärem Wurzelbaum $\rightarrow$ *Preorder*-Durchlauf des Baumes

Beispiel: Tiefensuche

Startknoten 1



Anmerkung: Reihenfolge bei der Auswahl der Nachbarn muss festgelegt werden.

Tiefensuche

- Anzahl der unbesuchten Knoten nimmt bei jedem Schritt um 1 ab.
- Graph zusammenhängend $\Rightarrow$ alle Knoten werden genau einmal besucht.
  - Um jeden Knoten eines nicht zusammenhängenden Graphen zu besuchen, muss der Algorithmus auf jede Zusammenhangskomponente angewendet werden.

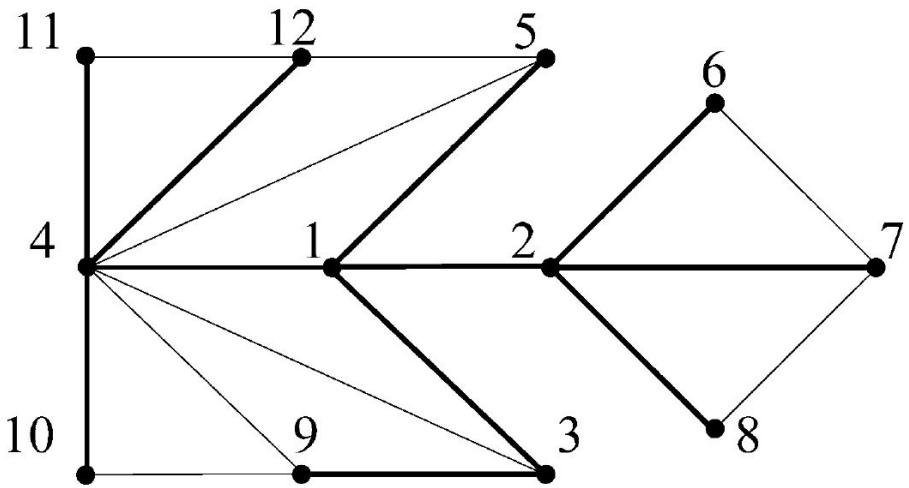
Markieren wir den Weg (die verwendeten Kanten), so erhalten wir einen Teilgraphen mit allen n Knoten und $n-1$ Kanten. Dieser Teilgraph ist ein Baum, ein sogenannter Spannbaum des Graphen G.

Anwendung: z. B. Ausgang aus einem Labyrinth finden

Breitensuche

Breitensuche – beginne bei beliebigem Knoten x von G:

1. Mache x zum aktuellen Knoten und gib diesem die Nummer 1.
2. Besuche alle zum aktuellen Knoten benachbarten Knoten und nummeriere diese fortlaufend durch, beginnend mit der nächsten freien Nummer.
3. Falls noch nicht alle Knoten besucht sind, mache Knoten mit der nächsten Nummer zum aktuellen Knoten und fahre mit Schritt 2 fort.



(Startknoten 1)

Breitensuche

- Ist der Graph zusammenhängend werden so alle Knoten genau einmal besucht.
  - Um jeden Knoten eines nicht zusammenhängenden Graphen zu besuchen, muss der Algorithmus auf jede Zusammenhangskomponente angewandt werden.
- Markieren wir den Weg (die verwendeten Kanten), so erhalten wir auch hier einen Spannbaum, der alle Knoten enthält.
  - Sieht jedoch völlig anders aus als jener der Tiefensuche!

Anwendung: z. B. kürzesten Weg zwischen zwei Knoten in unbewertetem Graphen finden

Anwendung: Dijkstra-Algorithmus

Ziel: Finde kürzesten Weg vom Knoten x zum Knoten y in einem *bewerteten* Graphen G (Anwendung z. B. bei Routing im Internet)

Voraussetzung: Alle Kantengewichte müssen positiv sein.

Vorgangsweise: Teile Knoten von G in drei Mengen:

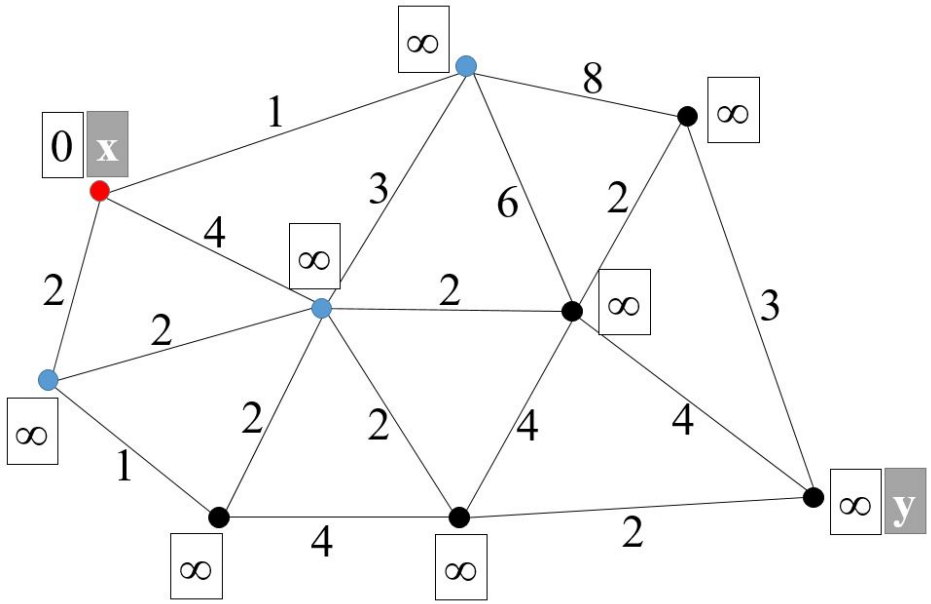
- **B**: schon besuchte Knoten, kürzeste Wege bekannt
- **R**: Nachbarn der Knoten aus B enthaltenen Knoten („Rand“ von B)
- **U**: noch nicht besuchte Knoten

Ausgangssituation: $B = \{x\}$, $R =$ Nachbarn von x

Anwendung: Dijkstra-Algorithmus

Algorithmus:

1. Zunächst besteht B nur aus dem Anfangsknoten. Seine Distanz $\alpha(x)$ wird auf 0, die Distanz aller anderen Knoten auf unendlich gesetzt.

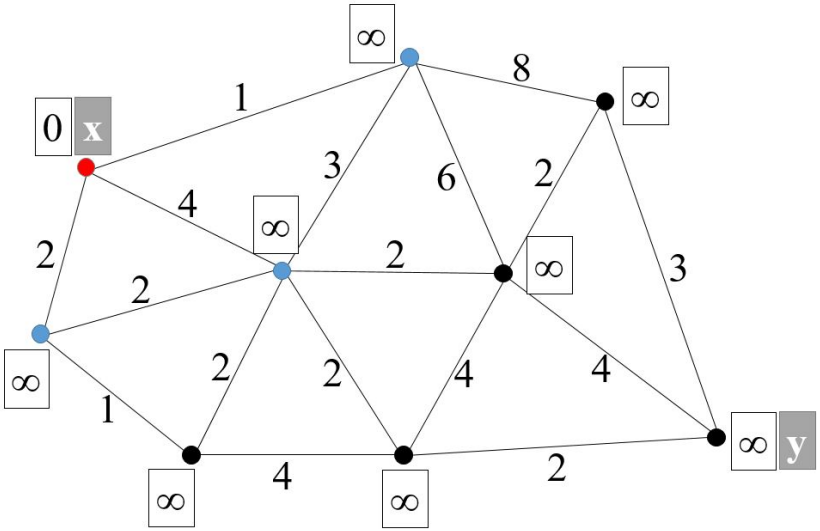


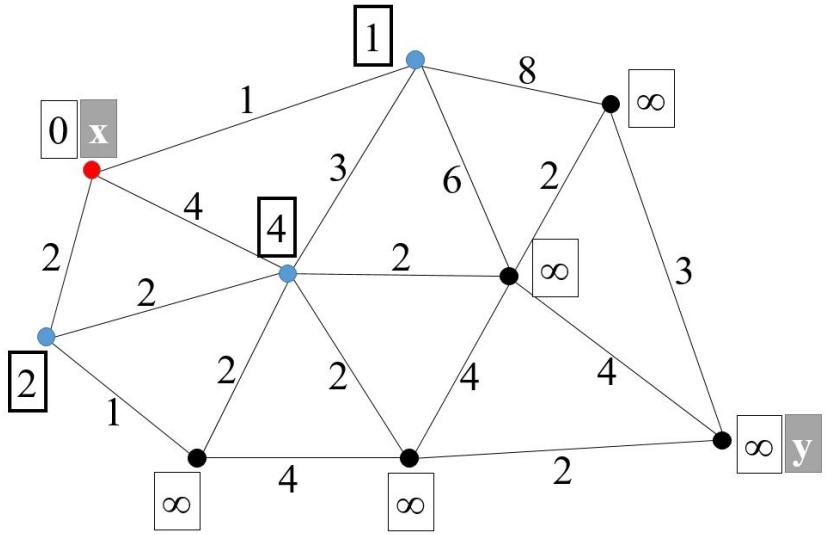
Anwendung: Dijkstra-Algorithmus

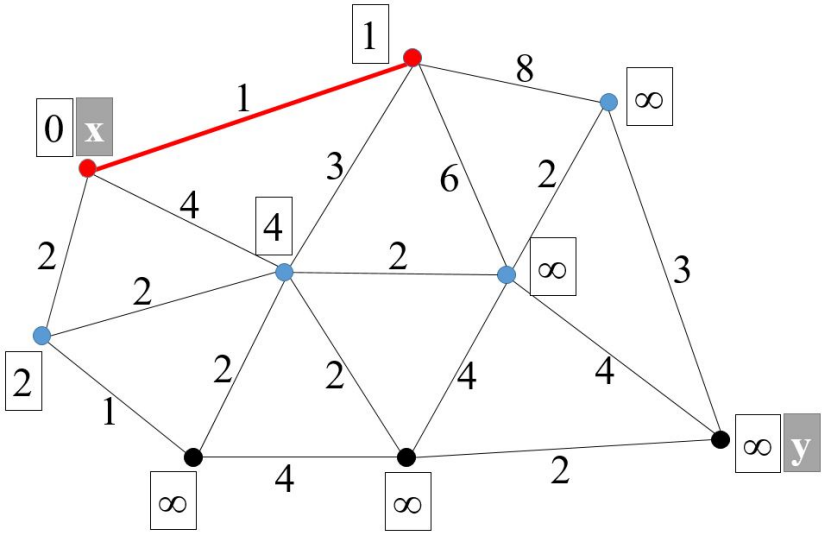
Algorithmus:

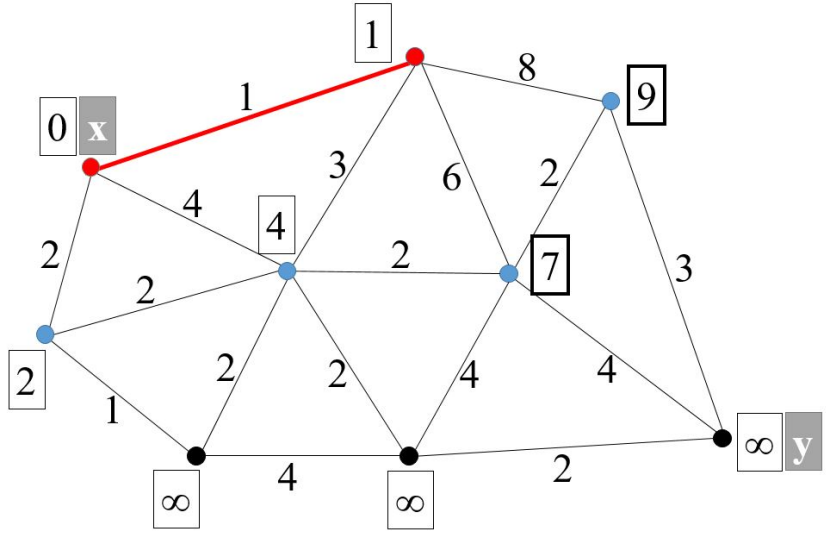
1. Zunächst besteht B nur aus dem Anfangsknoten. Seine Distanz $\alpha(x)$ wird auf 0, die Distanz aller anderen Knoten auf unendlich gesetzt.
2. Für jeden Knoten v_R in R wird $\alpha(v_B) + w(v_R)$ berechnet, wobei
   - $\alpha(v_B)$ die Distanz des mit v_R benachbarten Knotens aus B und
   - $w(v_R)$ die Bewertung der Verbindungskante dieser beiden Knoten ist.

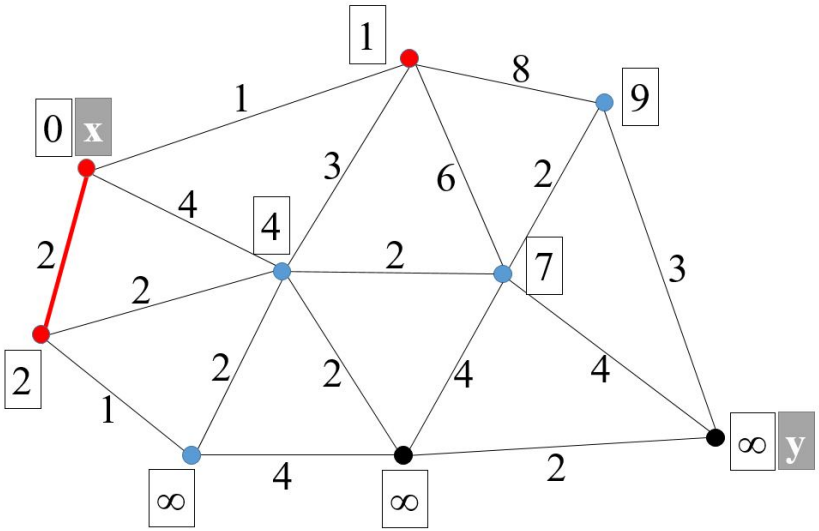
   (Ist v_R mit mehreren Knoten aus B benachbart, wird nur jene Verbindung berücksichtigt, bei der $\alpha(v_B) + w(v_R)$ minimal ist.)
3. Korrigiere die Distanz aller Knoten aus R, bei denen $\alpha(v_B) + w(v_R)$ kleiner als ihr aktueller Distanzwert ist, auf $\alpha(v_B) + w(v_R)$
4. Jener Knoten aus R, dessen (korrigierter) Distanzwert am geringsten ist, wird in B aufgenommen. Ist dieser Knoten nicht der Zielknoten y, fahre bei Punkt 2 fort; ansonsten ist der Algorithmus beendet.

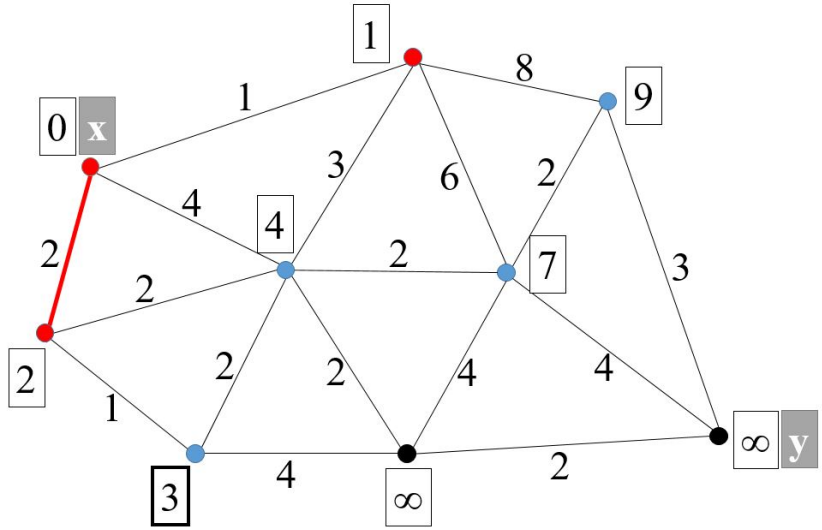

0 x
∞
∞
∞
∞
∞
∞
∞
∞
∞ y
1
8
4
3
6
2
2
2
2
3
2
2
4
4
1
4
2

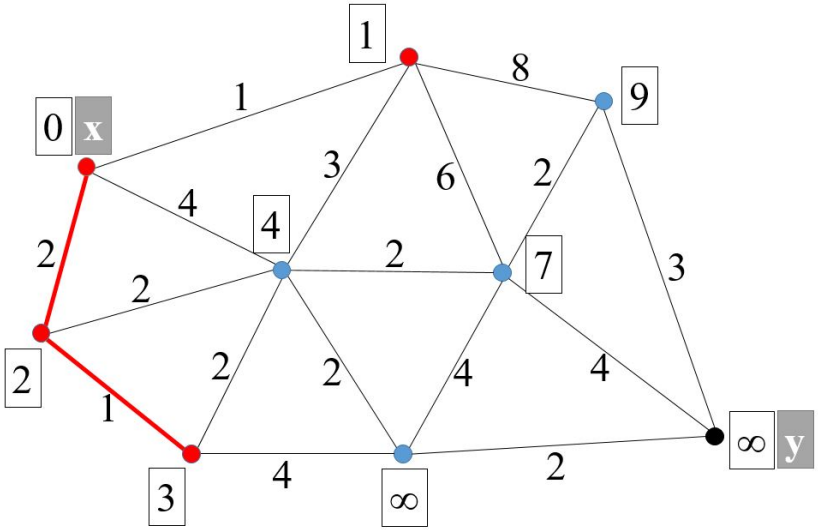

0 x
1
∞
4
∞
2
∞
∞
∞ y
1
8
4
3
6
2
2
2
2
3
2
2
4
4
1
4
2

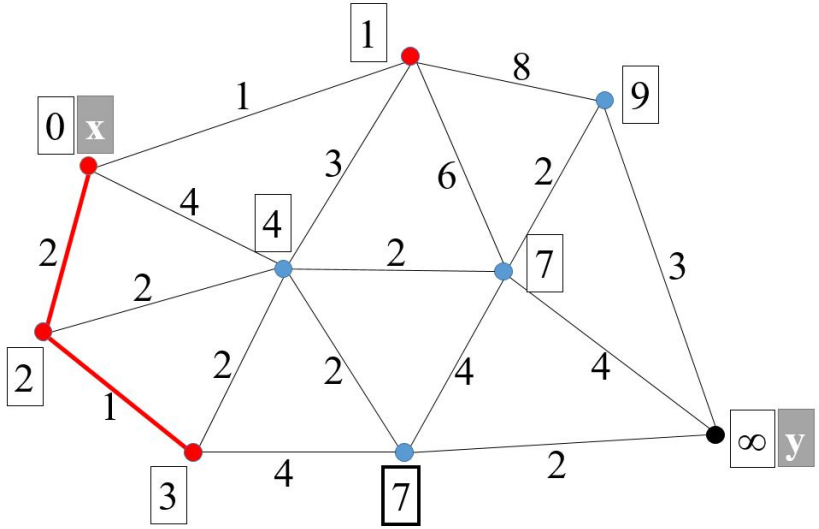

0 x
1
∞
4
∞
2
∞
∞
∞ y
1
8
4
3
6
2
2
2
2
3
2
2
4
4
1
4
2

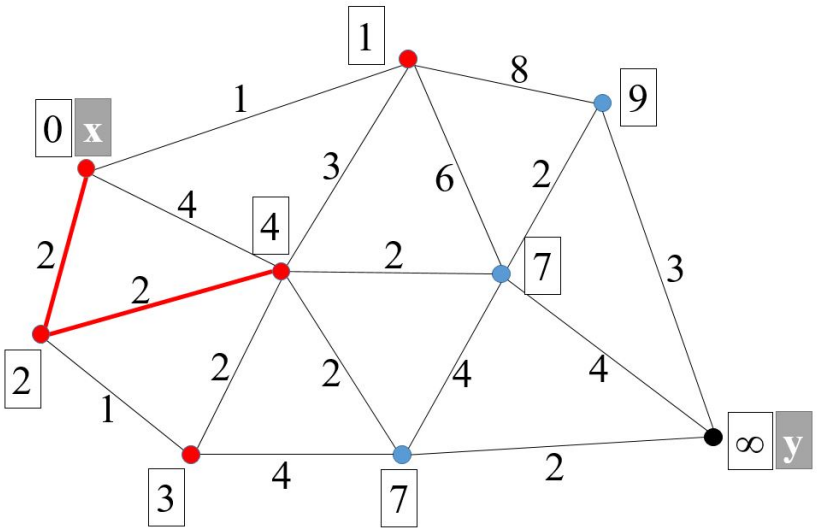

0 x
1
9
4
7
2
∞
∞
∞ y
1
8
4
3
6
2
2
2
2
3
2
2
4
4
1
4
2

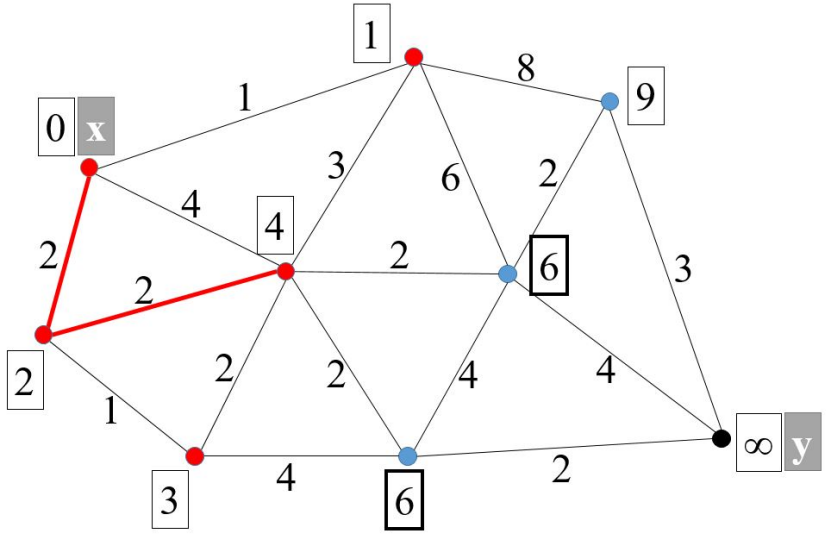

0 x
1
9
4
7
∞
∞
∞ y

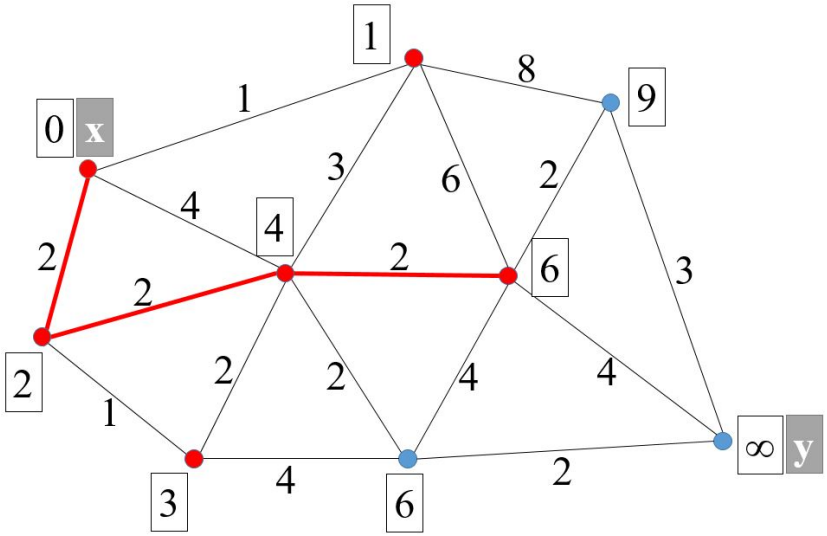

0 x
1
9
4
7
3
∞
∞
∞ y

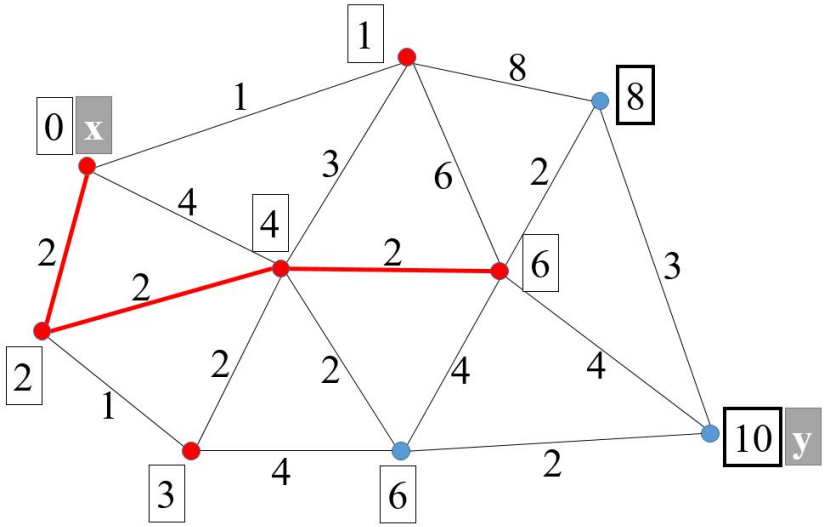

0 x
1
9
4
7
3
∞
∞
∞ y

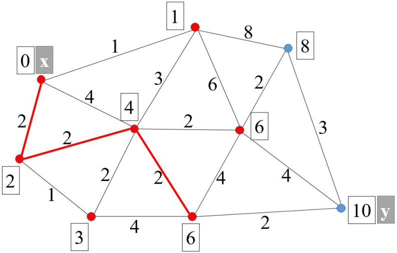

0 x
1
9
4
7
3
7
∞
∞ y

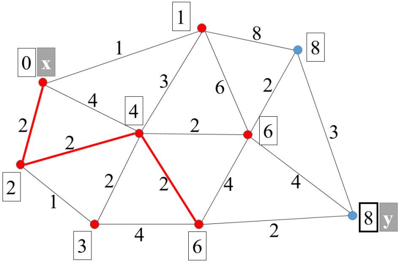

0 x
1
9
4
7
2
3
6
∞ y

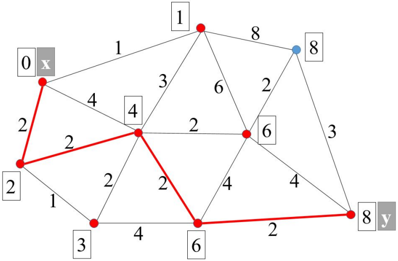

0 x
1
9
4
6
2
3
6
∞ y


0 x
1
9
4
6
2
3
6
∞ y


0 x
1
8
4
6
2
3
10 y


1
0 x
8
4
6
2
10 y
3
6
1
8
4
3
6
2
2
2
3
2
2
4
4
1
4
2


1
0 x
8
4
6
2
8 y
3
6
1
8
4
3
6
2
2
2
3
2
2
4
4
1
4
2


1
0 x
8
4
6
2
8 y
3
6
1
8
4
3
6
2
2
2
3
2
2
4
4
1
4
2

Gerichtete Graphen

Was ist ein gerichteter Graph?

Definition

Ein Digraph (für directed graph) oder gerichteter Graph G besteht aus einer Menge $V = V(G)$, den Knoten von G und aus einer Menge $E = E(G)$ von *geordneten* Paaren $k = [x, y]$, mit $x, y \in V$, den gerichteten Kanten von G.

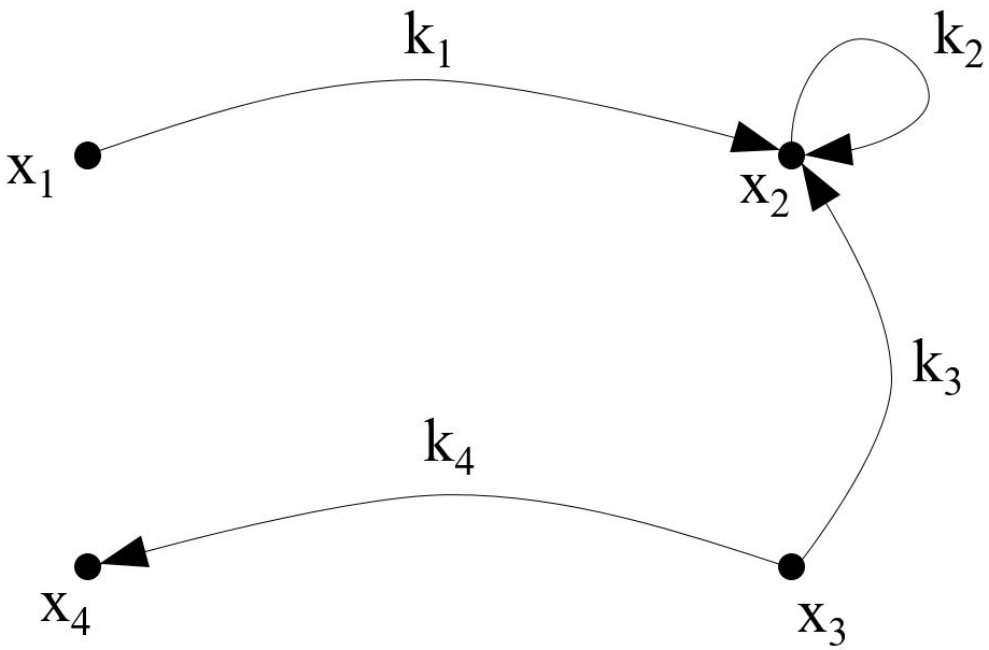
Die gerichteten Kanten heißen auch *Pfeile* oder *Bögen*. Die Richtung der Kante $k = [x, y]$ zeigt dabei von x nach y. x heißt Anfangspunkt und y Endpunkt der gerichteten Kante $[x, y]$.

$\rightarrow$ Einbahnstraßen, Flussdiagramme, Projektpläne, endl. Automaten, ...

Darstellung

Adjazenzmatrix hat $a_{ij} = 1$, wenn es einen Bogen *von i nach j* gibt.

- Keine symmetrische Adjazenzmatrix mehr
- Schlingen zählen nur einfach



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

ZH gerichtete Graphen – Relationen

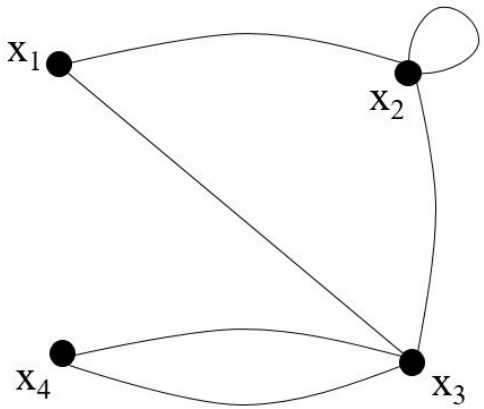
- Ein gerichteter Graph G ist eine Relation auf der Menge der Knoten:

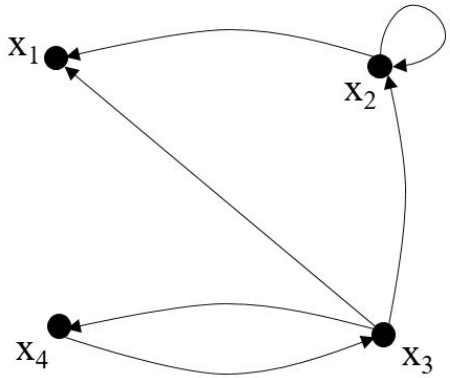
$$x\,R\,y \quad \Leftrightarrow \quad [x, y] \in E(G)$$

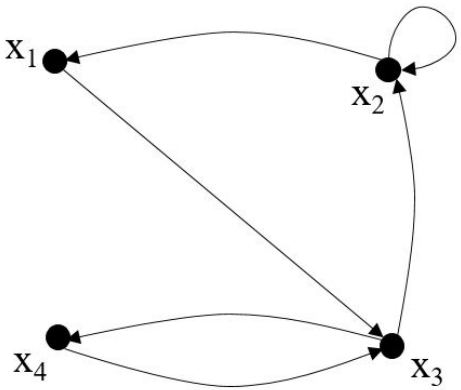
- Umgekehrt: Jede Relation auf einer endlichen Menge stellt einen gerichteten Graphen dar.

Zusammenhang

- Den **Schatten** eines (gerichteten) Graphen erhalten wir, indem wir die gerichteten Kanten des Graphen durch ungerichtete Kanten ersetzen, wobei zwischen zwei Knoten maximal eine Kante existiert.
- Ein gerichteter Graph G heißt **schwach zusammenhängend**, wenn der zugrunde liegende Schatten des Graphs zusammenhängend ist.
- G heißt **stark zusammenhängend**, wenn je zwei Knoten durch einen gerichteten Weg verbunden sind, wenn also jeder Knoten von jedem erreichbar ist.







Länge von Kantenfolgen

- Die Adjazenzmatrix gibt an, von welchen Knoten x aus man welche Knoten y *in einem Schritt* erreichen kann. Sie zeigt also die Gesamtheit der Kantenfolgen der Länge 1 an.
- Die quadrierte Adjazenzmatrix gibt an, wieviele Kantenfolgen der Länge 2 es vom Knoten x zum Knoten y gibt.
- **Allgemein:** Die n-te Potenz der Adjazenzmatrix gibt an, wieviele Kantenfolgen der Länge n es von Knoten x zu Knoten y gibt.

$$A^0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad A^1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \qquad A^2 = \begin{pmatrix} 2 & 2 & 0 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \end{pmatrix}$$

Erreichbarkeit

Um zu prüfen, ob jeder der n Knoten eines Graphen von jedem anderen aus *erreichbar* ist, kann die **Summe der** 0**-ten bis** $(n-1)$**-ten Potenz** der Adjazenzmatrix A gebildet werden.

- Werte > 0 können durch 1 ersetzt werden (für die Erreichbarkeit macht es keinen Unterschied, ob es zwischen zwei Knoten 1 oder 20 Kantenfolgen gibt).
- Enthält die Summenmatrix nur 1-Einträge, so ist jeder Knoten von jedem aus erreichbar.
- Die Summation kann sofort abgebrochen werden, sobald die aktuelle Summe nur noch 1 enthält.

Frage: Warum reicht es, die Summe bis zur $(n-1)$-ten Potenz zu bilden?

Beispiel: Erreichbarkeit

$$B_1 = A^0 + A^1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

$$B_2 = B_1 + A^2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 & 0 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Knotengrad in gerichteten Graphen

Eingangsgrad/indegree $d^-(x)$ von x:

- Anzahl der Bögen, die in x enden
- Entspricht der *Spalten*summe der Adjazenzmatrix

Ausgangsgrad/outdegree $d^+(x)$ von x:

- Anzahl der Bögen, die in x beginnen
- Entspricht der *Zeilen*summe der Adjazenzmatrix

x heißt Quelle, wenn $d^-(x) = 0$
x heißt Senke, wenn $d^+(x) = 0$

Satz

In jedem gerichteten Graphen $G = (V, E)$ *gilt*

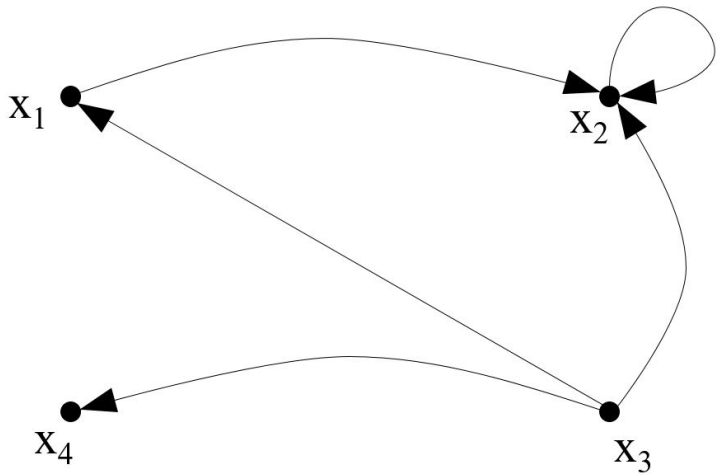
$$\sum_{x \in V} d^+(x) = \sum_{x \in V} d^-(x) = |E|.$$

Knotengrad in gerichteten Graphen

Satz

In einem gerichteten Graphen existiert genau dann eine **geschlossene Eulersche Linie** *(Eulerkreis), wenn für jeden seiner Knoten gilt:* $indegree = outdegree$.

Beispiel: Knotengrade



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

	x_1	x_2	x_3	x_4
d^+	1	1	3	0
d^-	1	3	0	1

Topologisch sortierte Graphen

Definition

Der gerichtete Graph G mit Knotenmenge $\{x_1, x_2, ..., x_n\}$ heißt topologisch sortiert, wenn für alle Knoten x_i gilt: Ist $[x_k, x_i]$ ein Bogen, so ist $k < i$.

...anders formuliert: In einem topologisch sortierten Graphen sind die Knoten so durchnummeriert, dass alle Vorgänger eines Knotens eine kleinere Nummer haben als der Knoten selbst.

Azyklische Graphen

Definition

Ein gerichteter Graph, in dem es keinen gerichteten Kreis gibt, heißt azyklischer gerichteter Graph.

Satz

In jedem azyklischen gerichteten Graphen gibt es mindestens eine Quelle und eine Senke.

Topologische Sortierung – azyklischer Graph

Azyklische gerichteten Graphen haben eine interessante Eigenschaft, die besonders bei der Abarbeitung der Knoten von Bedeutung ist:

- Die Reihenfolge der Abarbeitung kann so erfolgen, dass beim Besuch des Knotens x zuvor schon alle Knoten besucht worden sind, von denen aus ein Weg nach x führt.

Satz

Ein gerichteter Graph besitzt genau dann eine topologische Sortierung, wenn er azyklisch ist.

Topologische Sortierung – azyklischer Graph

Rekursiver Algorithmus zur topologischen Sortierung:

Nummeriere die Knoten des azyklischen gerichteten Graphen G

1. Suche eine Quelle x von G und nummeriere diese mit den nächsten freien Nummer.
2. Falls x nicht der einzige Knoten von G ist:
   - Entferne x und die von x ausgehenden Bögen $\rightarrow$ es entsteht der Graph G'.
   - *Nummeriere die Knoten des azyklischen gerichteten Graphen G'.*
3. Sonst ist die Nummerierung beendet.

$\rightarrow$ Wurden bei Beendigung des Algorithmus alle Knoten nummeriert, so ist der Graph azyklisch; findet man keine Quelle mehr, obwohl noch Knoten ohne Nummer vorhanden sind, so ist der Graph zyklisch (Kreis gefunden!)
$\rightarrow$ Ist der Graph azyklisch, so entspricht die Nummerierung einer topologischen Sortierung.

Bewerteter gerichteter Graph

Definition

Ein gerichteter Graph heißt bewertet, wenn jeder Kante $[x, y]$ ein Gewicht $w(x, y) \in \mathbb{R}$ zugeordnet ist.

Beispiele

- Transportkapazitäten von Pipelines
- Entfernungen in Routenplänen
- Zeitdauern in Projektplänen
- ...

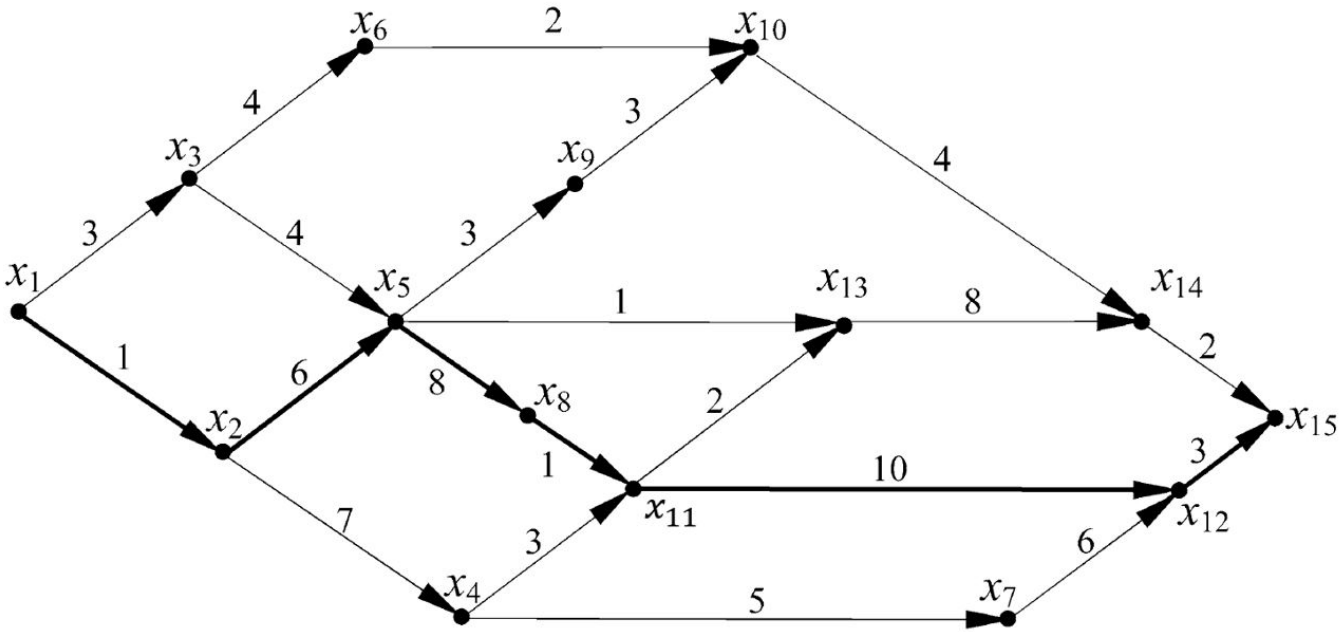
Beispiel: Projektplan

Der Plan für die Durchführung eines Projekts stellt einen bewerteten gerichteten Graphen dar:

- Ausgehend von einer Quelle (z. B. Auftragserteilung) wird das Projekt in verschiedene Module und Teilmodule unterteilt. Zwischen einzelnen Modulen bestehen Abhängigkeiten: Manche Module können erst gestartet werden, wenn andere beendet sind oder einen definierten Status erreicht haben.
- Die Bögen sind die Tätigkeiten/Abhängigkeiten in dem Projekt
- Die Bewertung eines Bogens stellt die Zeitdauer für die Durchführung der Tätigkeit dar.
- Die Knoten bezeichnen die Meilensteine des Projekts.

Beispiel: Projektplan

- Wir betrachten den Fall, dass es genau eine Quelle (Projektstart) und eine Senke (Projektende) in dem Graphen gibt.
- Der Quelle ordnen wir den Zeitpunkt 0 zu.
- **Gesucht**: frühester Zeitpunkt T, an dem die Senke erreicht werden kann ($\rightarrow$ Abschätzung des gesamten Zeitaufwands des Projekts)



Längster Weg in azyklischem Graphen

Beachten Sie:

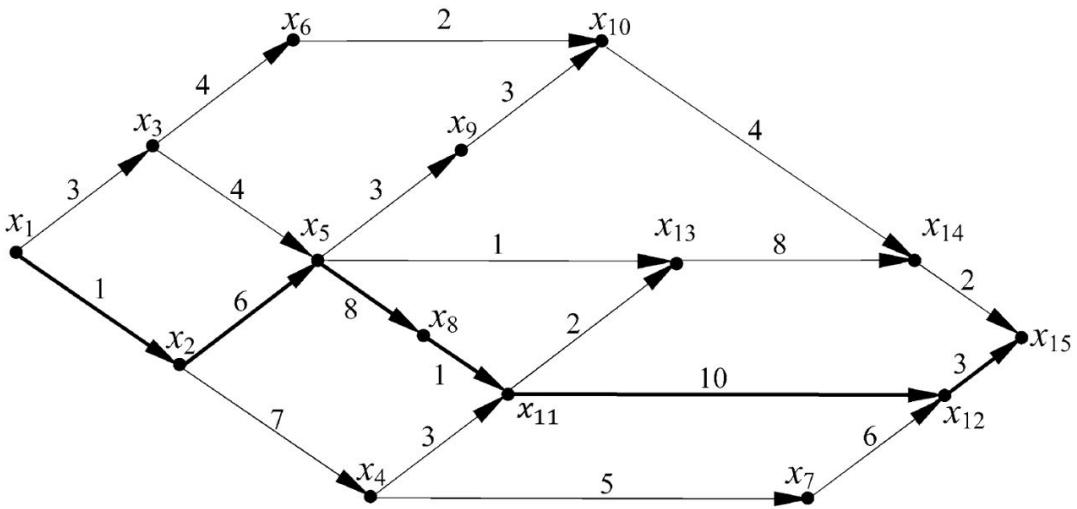
- Graph des Projektplans kann keinen Kreis enthalten.
- Da jede einzelne Aktivität durchgeführt werden muss (also *jeder* Weg des Graphen durchlaufen werden muss), stellt sich folgendes Problem:

  Finde den *längsten Weg* in einem azyklischen Graphen

Längster Weg in azyklischem Graphen

Gesucht: Alle $L(x_i) :=$ größte Länge eines gerichteten Weges von der Quelle bis $x_i =$ frühester Zeitpunkt des Erreichens von Meilenstein x_i
Voraussetzung: Topologische Sortierung (azyklisch!)

1. $L(x_i) := 0 \; \forall i$; $i := 1$
2. Besuche den Knoten x_i und untersuche alle seine Nachfolger y: $L(y) := \max\{L(x_i) + w(x_i, y); L(y)\}$
3. Ist $i + 1 = n \Rightarrow$ fertig; ansonsten $i \leftarrow i + 1$ und gehe zu Schritt 2



Längster Weg in azyklischem Graphen

- Algorithmus funktioniert, weil bei Besuch des Knotens x_i der längste Weg $L(x_i)$ dorthin schon endgültig berechnet ist: Wegen der topologischen Sortierung führt von späteren Knoten kein Bogen mehr nach x_i zurück, so dass $L(x_i)$ nicht mehr verändert werden kann.
- Damit ist $L(x_i) + w(x_i, y)$ der längste Weg nach y, der über x_i führt.
- Da im Lauf der Zeit sämtliche Vorgänger von y berücksichtigt werden, ist irgendwann auch der längste Weg nach y gefunden.
- Gesuchte minimale Projektdauer $T = L(x_n)$.
  - Im Beispiel: $L(x_{15}) = 29$