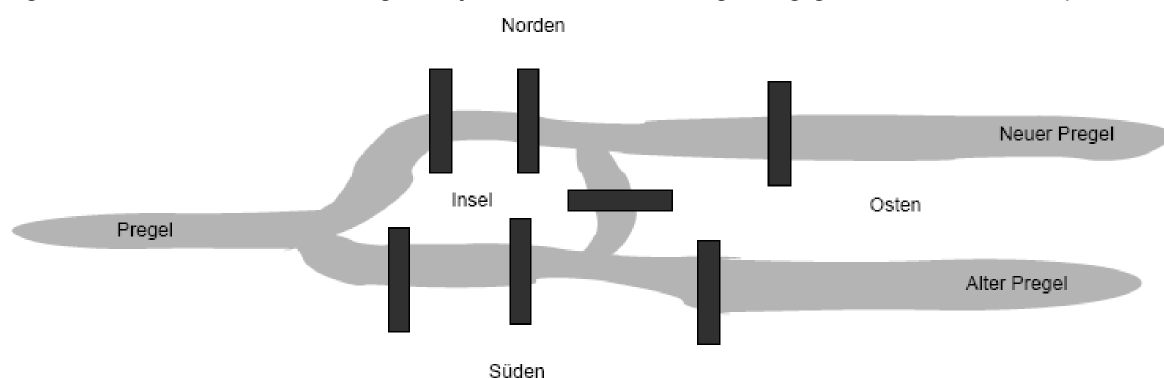
9. Graphentheorie

9.1. Worum geht es?

9.1.1. Historische Einleitung

Der Beginn der Graphentheorie wird allgemein mit dem Namen **Leonard Euler** und dem Jahr 1736 verbunden, als dieser eine Lösung für das **Königsberger Brückenproblem** angab: Gibt es einen Rundweg, der jede Brücke in Königsberg genau einmal überquert?



Eulers Antwort war Nein, und wie so oft in der Wissenschaft erwies sich diese negative Antwort (und ihr systematischer Beweis) fruchtbarer als eine positive Antwort. Eulers Arbeiten begründeten ein neues Wissensgebiet der Mathematik, die **Graphentheorie**.

9.1.2. Warum InformatikerInnen Graphen brauchen

Heute findet die Graphentheorie Anwendung in einer Vielzahl von Aufgaben, die in der Regel mit Netzen (Straßen, Elektrizität, Internet, ...) oder anderen graphartigen Strukturen zu tun haben:

- Routenplanung (gibt es einen Weg aus dem Labyrinth, was ist der kürzeste Weg (Kap. 9.4.2) von A nach B?)
- Projektplanung, Critical Path (bei welchen Teilaufgaben führt eine Verzögerung sofort zur Verzögerung des ganzen Projektes?)
- Klassendiagramme im objektorientierten Programmentwurf.

Eine spezielle Unterklasse von Graphen sind Bäume (Kap. 9.3), und diese spielen eine überragende Bedeutung in der Informatik:

- Wie erreicht man eine effiziente Datenhaltung in einer Bibliothek, einem Lexikon, einem Datenbestand, ..., derart, dass die Suche schnell geht? Die Antwort darauf sind **binäre Suchbäume** (Kap. 9.3.1).
- Wie übermittelt man eine Nachricht über Alphabet A möglichst komprimiert von Sender an Empfänger? Eine Antwort darauf lautet **Huffman-Code** (Kap. 9.3.2), und dieser baut auf einem Binärbaum auf. Der Huffman-Code ist Teil des **JPEG**-Algorithmus zur Bildkomprimierung.

Alle diese Probleme berühren wichtige Tätigkeitsfelder der Informatik.

9.2. Graphen

Lit.: [Hartmann, S. 202-224], [Stingl, S. 241-258]

Am Anfang der Graphentheorie müssen wir (leider) ein wenig "Vokabular pauken":

Def D 9-1: Graph, Knoten, Kante, Pfeil, Digraph

Ein (ungerichteter) **Graph** G besteht aus einer nichtleeren Menge $N=\{n_1,n_2,...\}$, den **Knoten** (engl. 'nodes'), und aus einer Menge K von ungeordneten Paaren $k = \{x,y\}$ mit $x,y \in N$, den **Kanten** von G. Man schreibt G = (N,K).

Jede Kante $k = \{n_i,n_j\}$ kommt höchstens einmal vor und für jede Kante k ist $n_i \neq n_j$.

Ein **Digraph** (von engl. 'directed graph', gerichteter Graph) hat als Kanten geordnete Paare $k = [n_i,n_j]$ = [Anfangsknoten, Endknoten]. Die Kanten nennt man dann **Pfeile**.

Jede Kante $k = [n_i,n_j]$ kommt höchstens einmal vor und für jede Kante k ist $n_i \neq n_j$.

Def D 9-2: Multigraphen

Ein Objekt ohne den letzten Satz aus Def D 9-1 heißt **Multigraph**. Ein Multigraph kann Kanten k=[x,y] mit x=y, sog. **Schlingen**, enthalten. Ferner kann es eine Kante von x nach y mehrfach geben (**parallele Kanten**). Die Menge K wird dann zu einer Multimenge, in der bestimmte Objekte (wie die Kante von x nach y) mit der Multiplizität m auftreten.

Im Digraphen (Pfeile statt Kanten), sind Pfeile nur dann parallel, wenn auch die Durchlaufrichtung gleich ist.

Ein Multigraph ist ein Graph, wenn er keine Schlingen und keine parallelen Kanten besitzt.

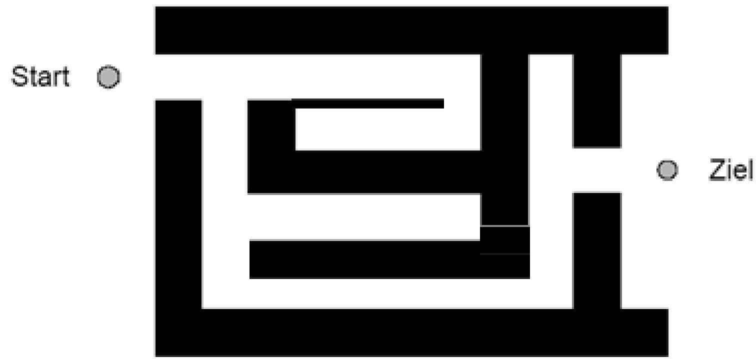
Anmerkungen:

- "ungerichtet" heißt: Die Kanten haben keine Richtung. Beim *gerichteten Graphen* sind die Kanten = "Einbahnstraßen".
- Beachte: Ein Digraph ist NICHT ein spezieller Graph, sondern ein (leicht) anderes Objekt, bei dem die Kanten durch Pfeile ersetzt sind. (s. auch Anmerkung unten, nach Def D 9-10)
- Ein Graph muss nicht zusammenhängend sein (es kann Knoten geben, die nicht durch eine Kantenfolge verbunden sind).
- Es kann auch Knoten ohne Kanten geben.
- Es gibt aber keine Kanten, die nicht an einem Knoten des Graphen anfangen oder enden.

Beispiele in Vorlesung!

Übung: Übersetzen Sie das Königsberger Brückenproblem in einen Multigraphen, in dem die Stadtteile die Knoten und die Brücken die Kanten sind. Schreiben Sie die Mengen N und K auf. Ist der Multigraph ein Graph?

Übung: Übersetzen Sie nachfolgendes Labyrinth in einen Graphen. Was identifiziert man zweckmäßigerweise mit den Knoten, was mit den Kanten?



Def D 9-3: vollständiger Graph, Teilgraph

Zwei Knoten eines Graphen heißen **benachbart**, wenn sie durch eine Kante verbunden sind. Ein Graph G = (N,K) heißt **vollständiger Graph**, wenn je zwei Knoten aus G auch benachbart sind. Ein Graph G' = (N',K') heißt **Teilgraph** von G, wenn $N' \subset N$ und $K' \subset K$ ist.

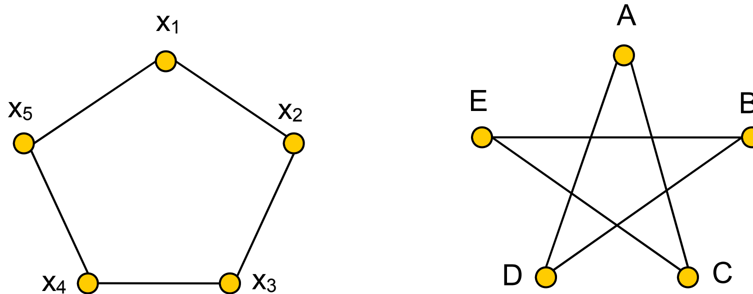
Beispiele in Vorlesung!

Def D 9-4 Isomorphie

Zwei Graphen G = (N,K) und G' = (N',K') heißen isomorph, wenn es eine bijektive Abbildung φ: N→N' gibt, so dass für alle $x,y \in N$ gilt:

$$[x,y] \in K \quad \Leftrightarrow \quad [\varphi(x), \varphi(y)] \in K'$$

Beispiel:



Mit der bijektiven Abbildung

x	x_1	x_2	x_3	x_4	x_5
$\varphi(x)$	A	C	E	B	D

stellen wir eine Isomorphie her, denn die Kanten aus K übersetzen sich in

(K)	$[x_1,x_2]$	$[x_2,x_3]$	$[x_3,x_4]$	$[x_4,x_5]$	$[x_5,x_1]$
(K')	[A,C]	[C,E]	[E,B]	[B,D]	[D,A]

und man überzeugt sich, dass dies auch genau alle Kanten von K' sind.

Anmerkungen:

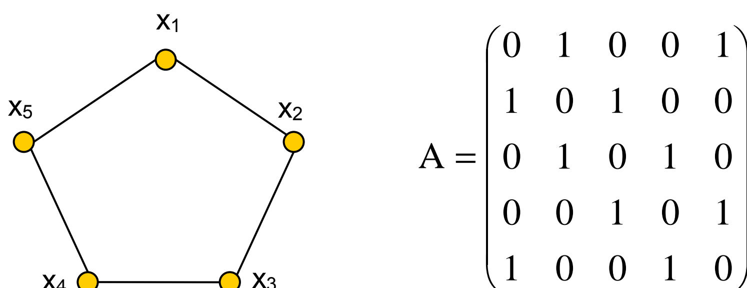
- Wieso ist Isomorphie wichtig? – Weil die Graphentheorie so aufgebaut ist, dass alle graphentheoretischen Eigenschaften, die für einen Graphen G gelten, auch für den isomorphen Graphen G' gelten. Damit kann man sich manchmal über den Beweis der Isomorphie viel Arbeit sparen.
- Es ist einfach, zu einem gegebenen Graphen G einen isomorphen Graphen zu konstruieren. Umgekehrt kann es sehr schwer sein, von zwei gegebenen Graphen die Isomorphie festzustellen. (Im obigen Beispiel sieht man die Isomorphie vielleicht auch nicht auf Anhieb, bei größeren Graphen ist es noch viel schwieriger.)

Repräsentation von Graphen im Computer:

Def D 9-5: Adjazenzmatrix

Die Knoten eines Graphen / Multigraphen werden von 1 bis n durchnummeriert. In der **Adjazenzmatrix** $A = (a_{ij})$, i=1,...,n, j=1,...,n ist a_{ij}=P, wenn von i nach j genau P Kanten (bzw. P Pfeile beim Multi-Digraphen) existieren, sonst 0.

Beispiel:



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

Anmerkungen

- Für "echte" Graphen sind 1 und 0 die einzig möglichen Werte.
- Die Adjazenzmatrix für ungerichtete Graphen ist immer **symmetrisch**. Bei Digraphen ist sie im allgemeinen nicht symmetrisch
- Die Hauptdiagonale der Adjazenzmatrix eines Graphen enthält nur "0"en.
- Für große Graphen, die nur spärlich verbunden sind (z.B. Ortsverbindungen im Routenplaner) kann die Adjazenzmatrix unhandlich groß werden. Dann empfehlen sich **Adjazenzlisten**.

Def D 9-6 Grad eines Knoten

Ist x ein Knoten eines Graphen, so ist der **Grad von x** die Anzahl der Kanten, die in ihm ansetzen (Schlingen zählen 2). Wir bezeichnen den Grad von x mit **d(x)**.

Bei **Digraphen** unterscheidet man den **Ausgangsgrad** $\mathbf{d^+(x)}$, die Anzahl der Pfeile, die von x ausgehen, und den **Eingangsgrad** $\mathbf{d^-(x)}$, die Anzahl der Pfeile, die in ihm enden.

Der Grad des Knoten i eines (ungerichteten) Graphen läßt sich über "Summe Zeile i" aus der Adjazenzmatrix ablesen.

Beim Digraphen ist

$$\sum_j a_{ij} = d^+(x_i) \qquad \text{(Zeilensumme = Ausgangsgrad)}$$

$$\sum_i a_{ij} = d^-(x_j) \qquad \text{(Spaltensumme = Eingangsgrad)}$$

Bsp. Internet: Die Struktur der Webseiten ist ein Digraph: Die Anzahl der Links, die von Website i ausgehen, ist ihr Ausgangsgrad, die Anzahl der Links, die auf Website i verweisen, ihr Eingangsgrad.

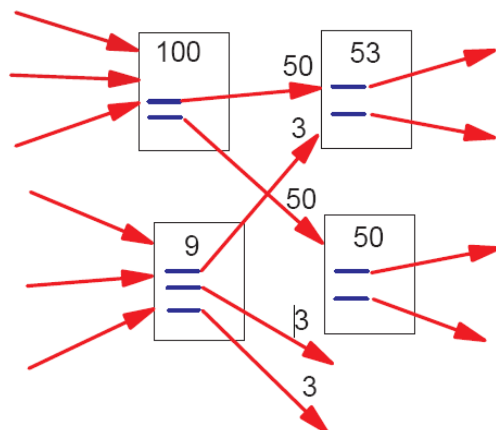


Abbildung 9-2: Ausschnitt des Webgraphen. Die Site "100" hat $d^-(x)=3$ und $d^+(x)=2$.

Wir werden in Kap. 13 das PageRank-Verfahren kennenlernen, das der Website x eine Bedeutung entsprechend ihres $d^-(x)$ zuweist. (Im obigen Bsp. sind die Zahlen der Sites fiktive Bedeutungswerte, die gleichmäßig auf die ausgehenden Pfeile transferiert werden.)

Satz S 9-1 Grad- und Kantenzahl

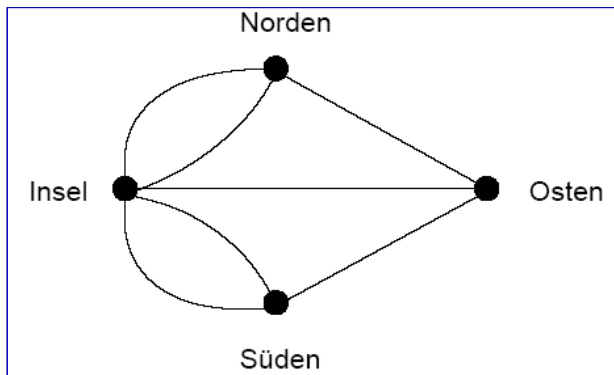
(1) In jedem Graphen G = (N,K) gilt: $\sum_{x \in N} d(x) = 2 \cdot |K|$.

(2) In jedem Digraphen G = (N,K) gilt: $\sum_{x \in N} d^+(x) = \sum_{x \in N} d^-(x) = |K|$.

(3) In jedem Graphen ist die Anzahl der Knoten ungeraden Grades gerade.

Bew. in Vorlesung. $|K|$ ist die Mächtigkeit der Menge K.

<u>Übung:</u> Bestimmen Sie den Grad aller Knoten im Königsberger Multigraph (links) und schreiben Sie die Adjazenzmatrix für den Königsberger und für den rechten Graphen auf:



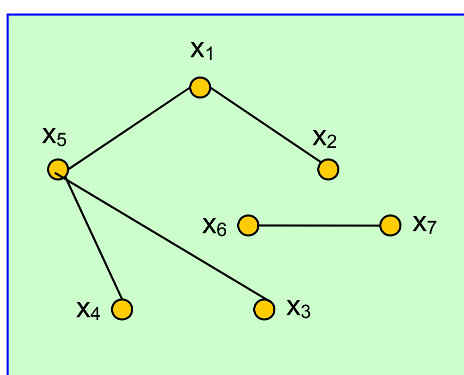


Abbildung 9-3: (links) Königsberger Graph, (rechts) ein weiterer Graph.

Wie ändert sich die Adjazenzmatrix, wenn Sie die Nummerierung von zwei Knoten, z.B. 6 und 2, vertauschen?

9.2.1. Wege in Graphen

Bei vielen Anwendung von Graphen sucht man Wege in Graphen mit bestimmten Eigenschaften. Typische Graphprobleme, die mit Wegen zusammenhängen, werden in Vorlesung besprochen.

Def D 9-7 Kantenfolge, Weg, Kreis, Länge

Ist G ein Graph mit Knoten x_i, so heißt eine Folge von Kanten, die x_1 mit x_n verbindet, eine **Kantenfolge von x_1 nach x_n**. Wir bezeichnen diese Folge $[x_1,x_2]$, $[x_2,x_3]$,..., $[x_{n-1},x_n]$ auch abkürzend mit $x_1x_2x_3...x_n$. Ist $x_1=x_n$, so heißt die Kantenfolge **geschlossen**.

Bei gerichteten Graphen müssen die Kanten in Pfeilrichtung durchlaufen werden.

Ein **Weg** ist eine Folge von **verschiedenen** Knoten $x_1x_2x_3...x_n$, für die $[x_i,x_{i+1}]\in K$ gilt mit i=1,...,n-1. Ausnahme Start/Endknoten: Es darf $x_1=x_n$ **für $n\geq 3$** gelten, dann nennt man den Weg einen **geschlossenen Weg** oder **Kreis**.

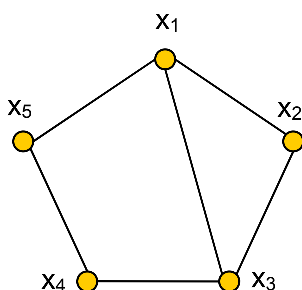
Die Anzahl der Kanten eines Weges (bzw. einer Kantenfolge) heißt **Länge des Weges** (bzw. der Kantenfolge).

Def D 9-8 Zusammenhang

Ein Graph G heißt **zusammenhängend**, wenn je zwei seiner Knoten durch Wege verbunden sind.

Anmerkungen:

- Ein Weg $x_1x_2x_3...x_n$ hat die Länge n-1. Ein Kreis $x_1x_2x_3...x_nx_1$ hat die Länge n, weil noch die Kante $[x_n,x_1]$ hinzukommt.
- zu „$n\geq 3$“: Den trivialen „Kreis“ der Länge 2, z.B. $x_1x_3x_1$, schließen wir aus, denn sonst hätte jeder Graph (mit nichtleerer Kantenmenge) lauter Kreise.



Beispiele:

1. Mögliche Kantenfolgen in nebenstehendem Graph sind

(a) $x_1x_2x_3x_1x_2x_3x_1x_5x_4$ oder
(b) $x_1x_2x_3x_1x_5$.

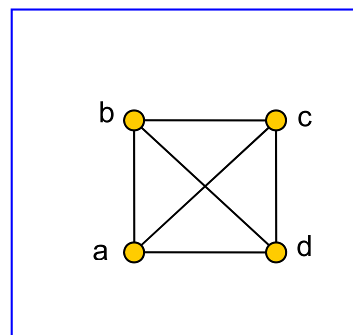
Bei (a) wird ein Kreis 2mal durchlaufen, (b) enthält einen Kreis. Beides sind aber keine Wege, da Knoten mehrfach auftauchen. Die möglichen Wege von x_1 nach x_4 sind:

$x_1x_5x_4$, $x_1x_3x_4$ oder $x_1x_2x_3x_4$.

2. Der rechte Graph aus Abbildung 9-3 ist nicht zusammenhängend. Er zerfällt in zwei Komponenten, seine sog. *Zusammenhangskomponenten*.

Übung: Im vollständigen 4er-Graphen stellt die Kantenfolge **[ab bc ca]** einen Kreis dar. Wir können diesen Kreis auch kurz durch **abca** notieren. Schreiben Sie alle Kreise in diesem Graphen auf! Auf wieviele Kreise kommen Sie insgesamt?

[Hinweis: Kreise, die lediglich Permutationen von anderen Kreisen sind, z.B. **bcab** oder **cbac** zu obigem Bsp. brauchen Sie nicht gesondert aufzuschreiben. Aber *überlegen* Sie trotzdem: Wieviele Kreise gibt es, wenn jede Permutation als eigener Kreis zählt?]



Im Praktikum wird besprochen, wie man in bestimmten Digraphen die Anzahl der Wege mit Länge n über Potenzen der Adjazenzmatrix erhält.

Def D 9-9 Euler-Zug

In einem Multi-Graphen G heißt eine Kantenfolge, die jede Kante genau einmal besucht und an ihren Ausgangspunkt zurückkehrt, ein **Euler-Zug**.

Anschaulich: Kann ich den (Multi-)Graphen in einem Zug zeichnen?

BEACHTE: Der Euler-Zug wird in der Regel KEIN Weg oder Kreis sein, denn offensichtlich müssen Knoten mit mehr als 2 Kanten mehrmals besucht werden. Dies ist leider bei [Brill01] falsch dargestellt.

Das Königsberger Brückenproblem reduziert sich also auf die Frage: Gibt es im Königsberger Multi-Graphen einen Euler-Zug?

Satz S 9-2 Satz von Euler

Ein Multigraph G enthält Euler-Züge genau dann, wenn jeder Knoten in G geraden Grad hat.

Damit ist das Königsberger Brückenproblem gelöst: Es gibt keinen Rundweg, denn alle Knoten haben ungeraden Grad.

Übung: Mit wieviel und welchen zusätzlichen Brücken können Sie das Königsberg-Problem lösbar machen? Geben Sie einen Euler-Zug an.

9.3. Bäume

[Stingl, S. 251f] [Hartmann, S. 206f]

Zunächst wieder ein wenig Vokabular:

Def D 9-10 Baum, Abstand

Ein Graph, in dem je zwei Knoten durch genau einen Weg verbunden sind, heißt **Baum**.

Der **Abstand** a(x,y) von zwei Knoten eines Baumes ist die Länge des (eindeutigen) Weges zwischen ihnen.

Def D 9-11 Wurzelbaum

Ein **Wurzelbaum** ist ein Digraph

a) bei dem der zugrundeliegende Graph ein Baum ist und
b) bei dem ein Knoten x_0 als Wurzel ausgewählt ist und jede Kante [x,y] durch einen Pfeil (x,y) ersetzt ist, der von der Wurzel wegweist.

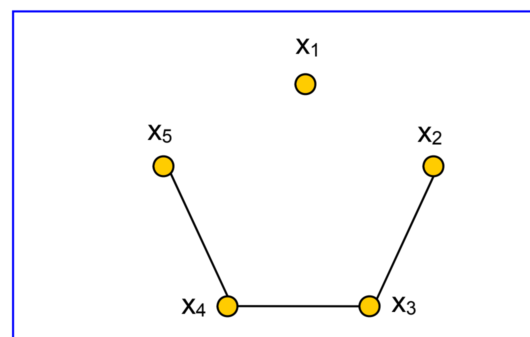
Def D 9-12 Binärbaum, regulärer Binärbaum

Ein **Binärbaum** ist ein Wurzelbaum, in dem jeder Knoten x höchstens zwei Nachfolger hat (Ausgangsgrad $d^+(x) \leq 2$). Hat jeder Knoten x entweder 0 oder 2 Nachfolger, so heißt der Baum **regulärer Binärbaum**.

Die Nachfolger eines Knoten x heißen **linker** und **rechter Sohn von x**. Die Binärbäume, die entstehen, wenn man den linken / rechten Sohn von x als Wurzel nimmt (und alle Vorgängerknoten wegläßt), nennt man **linker** und **rechter Teilbaum von x**.

Beachte: Die Baum-Definition Def D 9-10 gilt für Graphen, nicht für Digraphen. Ein Digraph D ist genau dann ein Baum, wenn Def D 9-10 für den D zugrundeliegenden Graphen gilt (!)

Zahlreiche Beispiele in Vorlesung!



Für Bäume gelten eine Reihe nützlicher Eigenschaften:

Satz S 9-3 Eigenschaften von Bäumen

Sei Graph G ein Baum

(1) Baum G ist ein zusammenhängender, kreisfreier Graph.
(2) Entfernt man irgendeine Kante aus einem Baum, so zerfällt er in zwei Zusammenhangskomponenten, die wieder Bäume sind, die **Teilbäume**.
(3) Ein Baum mit n Knoten hat genau n-1 Kanten.

Beweis in Vorlesung!

Wozu sind Bäume gut? Nachfolgend diskutieren wir zwei wichtige Anwendungen für Bäume:

9.3.1. Suchbäume

Wenn Sie in einem Datenbestand mit 100.000 Einträgen jeden Eintrag schnell wiederfinden wollen, ist ein Suchbaum die richtige Datenstruktur. Nehmen wir an, Ihre Daten besitzen einen sortierbaren Schlüssel. Ein Suchbaum entsteht durch folgenden Algorithmus:

Def D 9-13 Suchbaum

Ein Suchbaum ist ein Binärbaum, der nach folgender Vorschrift gebildet wird: Gegeben sei eine Menge von Objekten, von denen jedes einen sortierbaren Schlüssel s besitzt.

Trage Schlüssel s in den Baum ein:

Existiert Baum noch nicht, so erzeuge Wurzel und trage s als Wurzelschlüssel ein.
Sei s_0 der Wurzelschlüssel.

Sonst: Falls $s<s_0$: Trage Schlüssel s im linken Teilbaum der Wurzel ein.
Falls $s>s_0$: Trage Schlüssel s im rechten Teilbaum der Wurzel ein.

Beispiel: "Möchten Sie die Worte dieses Satzes selektieren und gut auffinden?"
Der Schlüssel sei das Wort selber, in alphabetischer Sortierung.

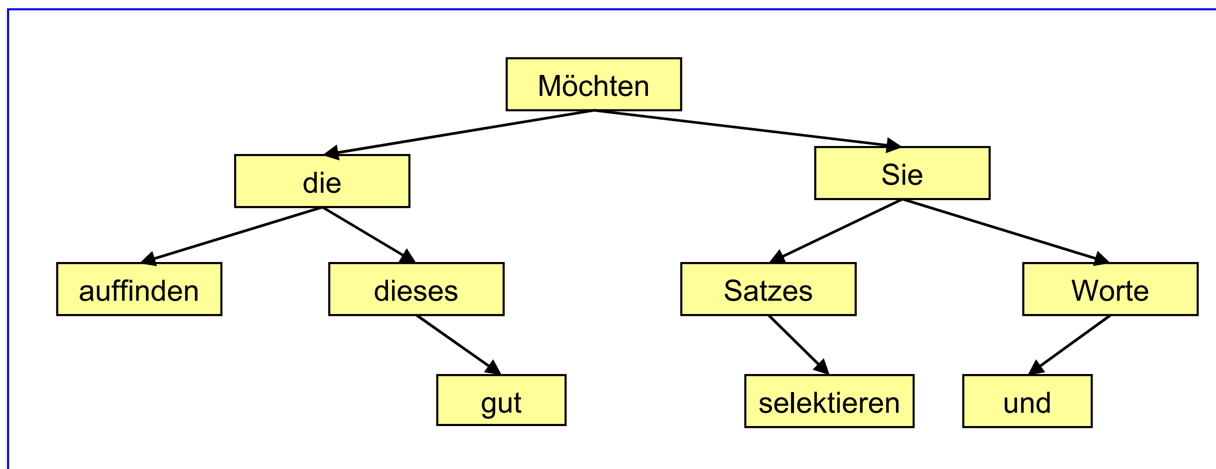


Abbildung 9-4: Aufbau eines Suchbaumes

Die Suche eines bestimmten Eintrages, z.B. "Satzes", geschieht durch folgende rekursive Suchvorschrift:

Suche s beginnend bei x:

Ist s gleich dem Schlüssel x, so liefere x zurück.
Falls $s<x$: Suche s beginnend beim linken Sohn von x
Falls $s>x$: Suche s beginnend beim rechten Sohn von x

Man braucht beim Wort "Satzes" also 3 Vergleiche. Wieviele kann man maximal speichern, wenn man bis zu 18 Vergleiche erlaubt?

Def D 9-14 Länge und Höhe eines Wurzelbaumes

Sei B ein Wurzelbaum. Der Maximalwert unter den Abständen (Def D 9-10) $a(x, x_0)$ der Knoten $x \in B$ zur Wurzel x_0 heißt **Länge L des Wurzelbaumes** (Abstand = Anzahl Kanten). Die Anzahl der Knoten des längsten Weges zur Wurzel x_0 heißt **Höhe H des Wurzelbaumes**.

Satz S 9-4 Knotenzahl in Binärbäumen

Sei B ein Binärbaum mit Höhe H. Dann enthält B maximial $N = 2^H - 1$ Knoten

Beispiele:

- Es gilt also nach Def D 9-14: H = L +1.
- Der Baum aus Abbildung 9-4 hat Höhe 4 und Länge 3
- Der vollständig besetzte Binärbaum der Länge 2 [zeichnen!] enthält $1+2+2^2=7=2^3-1$ Knoten.

Beweis in Vorlesung.

Damit folgt: Mit der Suchstrategie "Suche s beginnend bei x" in einem Baum der Höhe H brauchen wir maximal V=H Vergleiche. Bei 18 Vergleichen sind also maximal $2^{18}-1 \approx$ 262.000 Einträge speicherbar.

Der Suchaufwand ist also enorm verkürzt gegenüber einer "naiven Suche" in einer unsortierten Liste, die im Mittel 131.000 Vergleiche benötigt (klar?). Allgemein steigt der Suchaufwand im Binärbaum nur mit ld(N+1), wenn N die Zahl der Datensätze ist.

In Vorlesung: Ausblick auf **balancierte Bäume**.

9.3.2. Huffman-Code

Der Huffman-Code ist eine wichtige Anwendung für Binärbäume (Kompressionsverfahren).

Bitte aus [Hartmann04, S. 213-215] im Selbststudium erarbeiten!

9.4. Durchlaufen von Graphen

- evtl. Durchlaufen v. Graphen nur im Selbststudium, dafür MST genau ! -

Für viele Anwendungen von Graphen muss man diesen entlang seiner Kanten durchlaufen

1. Alle m Knoten einer Zusammenhangskomponente besuchen
    a. um festzustellen, ob der Graph zusammenhängend ist (Ist m=Gesamtzahl?)
    b. um alle Knoten der Zusammenhangskomponente genau einmal über die gegebenen Graphkanten zu erreichen (z.B. für Update-Vorgänge)
2. Kürzesten Weg von Knoten x nach Knoten y finden

Zur Erreichung von Ziel 1. gibt es zwei Varianten, die Tiefensuche und die Breitensuche. In beiden Fällen führen wir einen globalen Zähler `zahl` mit. Initial ist `zahl=1`. Für die Breitensuche haben wir noch einen weiteren Zähler `mark`, initial ist `mark=1`.

Durchlaufe den Graphen ab Knoten x in Tiefensuche:

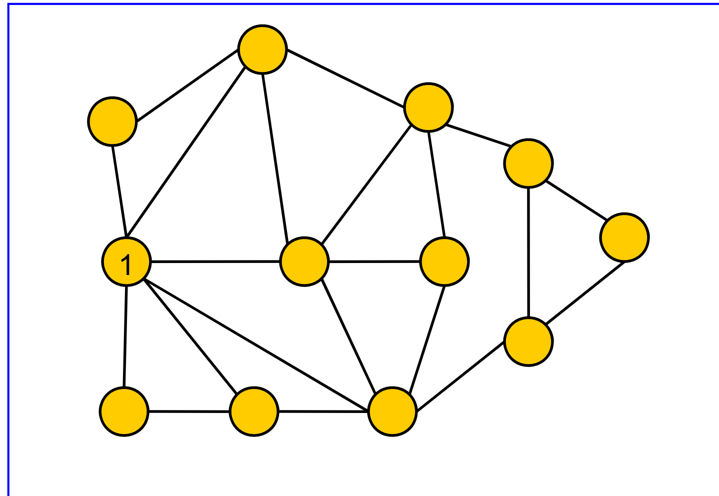
Markiere x als besucht, gebe ihm die Nr. `zahl` und erhöhe `zahl=zahl+1`.
Für alle noch nicht besuchten Knoten y, die zu x benachbart sind:
  Gehe zu y
  Durchlaufe den Graphen ab Knoten y in Tiefensuche

Durchlaufe den Graphen ab Knoten x in Breitensuche:

1. Mache x zum aktuellen Knoten und gebe ihm die Nr. `zahl`.
2. Besuche alle Nachbarn y des aktuellen Knoten, die noch keine Nummer tragen: Markiere Kante [x,y] als besucht, erhöhe `zahl=zahl+1` und gebe y die Nr. `zahl`.
3. Wenn `mark < zahl`: Setze `mark=mark+1`, mache Knoten `mark` zum aktuellen Knoten und weiter bei 2.

Wenn man alle besuchten Knoten und Kanten zusammennimmt, dann führt jedes dieser Verfahren zu einem Baum, der *alle* m Knoten dieser Zusammenhangskomponente besucht. Wir nennen einen solchen Baum einen *aufspannenden Baum*.

Übung: Entwickeln Sie den aufspannenden Baum ab Knoten 1 in Tiefensuche und in Breitensuche für folgenden Graphen:



[Hinweis: Wenn Sie mehrere Möglichkeiten haben, nehmen Sie den nächsten Knoten, von "9 Uhr" her gesehen im Uhrzeigersinn. (Damit wir besser die Lösung vergleichen können)]

9.4.1. Aufspannende Bäume, Algorithmus von Kruskal

Def D 9-15 Aufspannender Baum

Sei G ein zusammenhängender Graph mit N Knoten. Ein Teilgraph T von G, der ein Baum ist und der ebenfalls N Knoten enthält, heißt **aufspannender Baum** (engl. "spanning tree").

Beispiel: Gegeben sei ein Graph G mit N Rechnern, die über die Kanten des Graphes kommunizieren können. Wir wollen erstens wissen, (1) ob jeder Rechner mit jedem kommunizieren kann und (2) wollen wir überflüssige Doppelwege vermeiden. Wie finden wir heraus, welche Verbindungen wir ohne Schaden entfernen können?

Lösung: Beide Fragen beantwortet der aufspannende Baum T: Ist $m=N$ (d.h. die Tiefen- oder Breitensuche liefert Zähler `zahl`$=N$), so kann "jeder mit jedem". Zweitens können alle Kanten, die nicht zu T gehören, aus G entfernt werden, ohne die Kommunikationsfähigkeit des Netzes einzuschränken.

Ein aufspannender Baum ist also sparsam, in dem Sinne, dass er überflüssige Kanten vermeiden hilft. Nach Satz S 9-3 (3) hat jeder Baum mit N Komponenten gleich viele Kanten, nämlich N-1.

Aber im realen Leben ist nicht Kante gleich Kante. Manche Verbindungen sind länger, manche sind vielleicht teurer im Unterhalt. Oft geht es darum, den Kantensatz zu finden, der nach einem bestimmten Bewertungsschema minimale Kosten verursacht.

Def D 9-16 Bewerteter Graph

Ein Graph G(E,K,w) heißt **bewertet**, wenn jeder Kante $[x,y]$ ein Gewicht $w(x,y) \in \mathbf{R}$ zugeordnet ist.

In bewerteten Graphen ist die **Länge** (vgl. Def D 9-7) eines Weges die Summe seiner Kantengewichte.

In der Adjazenzmatrix (vgl. Def D 9-5) bewerteter Graphen schreibt man für jede existierende Kante statt der "1" das Gewicht $w(x,y)$.

Def D 9-17 Minimaler aufspannender Baum (MST)

Sei T ein aufspannender Baum für den bewerteten Graphen G. T heißt minimaler aufspannender Baum (engl.: minimum spanning tree = **MST**), wenn die Summe seiner Kantengewichte

$w(T) \equiv \sum_{k \in K(T)} w(k)$ minimal ist.

Die Aufgabe lautet nun: Wie findet man den MST?

Bloßes Probieren geht nicht, denn es gibt ja astronomisch viele aufspannende Bäume (genauer gesagt, beim vollständigen Graphen nach [Aig96] sogar N^{N-2} viele, eine wahrhaft riesige Zahl).

Glücklicherweise gibt es einen ganz einfachen Algorithmus, der das Problem löst

Satz S 9-5 Algorithmus von Kruskal

Für einen zusammenhängenden, bewerteten Graphen G(E,K,w) bestimmt der folgende Algorithmus einen minimal aufspannenden Baum T:

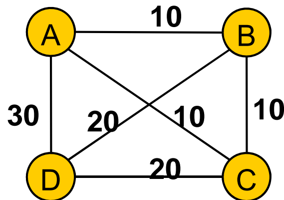
1. Setze T = {}, i=0.
2. Besuche unter den noch nicht besuchten Kanten diejenige mit kleinstem Gewicht. Falls diese einen Kreis in T schließt, verwerfe sie, ansonsten füge sie zu T hinzu und setze i=i+1.

3. Falls i=n-1, dann Stop. Ansonsten weiter bei Schritt 2.

Bew.: s. [Brill01, S. 236]

Beispiel in Vorlesung!

Beispiel, bei dem mehrere MSTs möglich sind



Kanten geordnet aufschreiben:

Übung: Finden Sie durch systematisches Vorgehen ALLE MSTs zu dem durch folgende Kostentabelle gegebenen bewerteten Graphen:

B	C	D	E	
20	20	10	40	A
	30	20	50	B
		30	50	C
			40	D

Exkurs: Der Kruskal-Algorithmus folgt einer sog. **"greedy"** Strategie. "greedy" heißt soviel wie gierig (gefräßig), und bedeutet, dass man beim schrittweisen Voranschreiten in Richtung eines bestimmten Optimalziels in jedem Schritt unter den vorhandenen Alternativen gierig die momentan (lokal) beste auswählt. Hier funktioniert es zwar, aber bei vielen anderen Problemen führt ein solches Vorgehen nicht zu einer optimalen Lösung.

Gegenbeispiel Rucksack-Problem: s. Vorlesung.

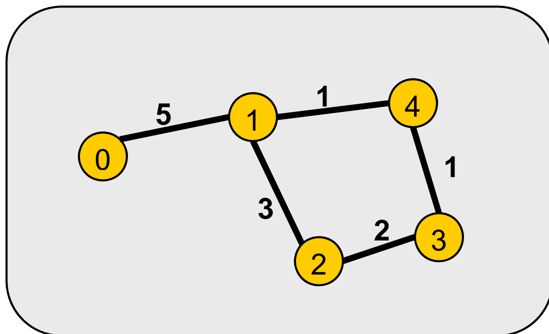
Greedy Strategien sind also in der Regel nicht optimal, aber meist mit geringem Aufwand zu realisieren. Sie liefern eine *untere Benchmark* bei der Entwicklung optimaler Strategien ("Wie weit komme ich mit einfachen Mitteln?")

9.4.2. Kürzeste Wege, Algorithmus von Dijkstra

Kürzeste Wege in komplexen Graphen zu finden ist *von enormer praktischer Bedeutung*: Welches ist das schnellste (oder das kostengünstigste) Routing für eine Nachricht im Internet? Welches ist der kostengünstigste Transportweg in einem Gasleitungsnetz? Die kürzeste Route zwischen zwei Städten A und B? Usw., usw.

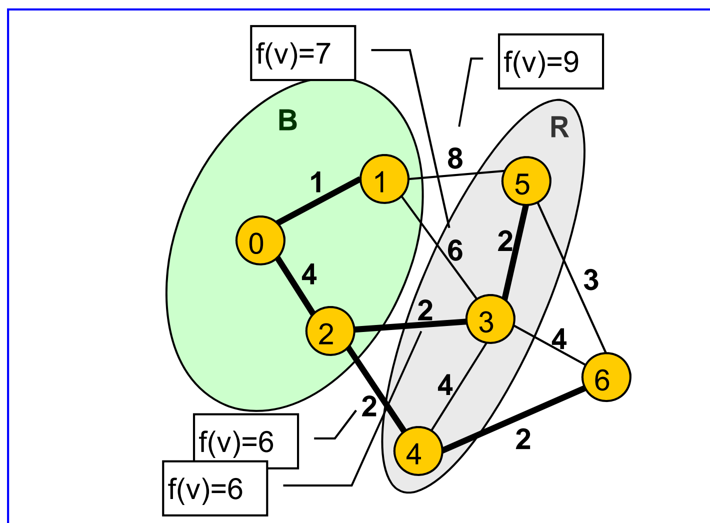
Auch für kürzeste Wege gibt es ein Verfahren, den berühmten **Algorithmus von Dijkstra**, mit dem sich für vorgegebenen Startknoten ein aufspannender Baum aller kürzesten Wege bestimmen läßt. Glücklicherweise braucht man auch hier nicht alle möglichen N^{N-2} aufspannenden Bäume durchzupropieren.

Übung: Haben wir nicht mit dem MST schon einen Algorithmus, der die kürzesten Wege aufzeigt? – Überlegen Sie anhand des folgenden bewerteten Beispielgraphen, wie der Baum KW0 aller kürzesten Wege ab Startknoten 0 und wie der MST aussieht!



Voraussetzung für den Algorithmus von Dijkstra: ein bewerteter Graph (kein Multigraph) mit ausschließlich **positiven** Bewertungen.

Der Algorithmus wird im Praktikum ausführlich besprochen und durchgeführt (dort allerdings leicht andere Variante). Die Grundidee zeigt nachfolgendes Bild, wird in Vorlesung näher erläutert:



Der Algorithmus von Dijkstra benutzt folgende Bestandteile: Sei **B** die Menge der besuchten Knoten, **R** der **Rand** um B herum. Jedem Knoten n wird im Laufe des Dijkstra-Algorithmus die **Länge l(n)** des kürzesten Weges zugewiesen und sein **Vorgänger-Knoten W(n)** auf diesem Weg zu ihm:

Satz S 9-6 Algorithmus von Dijkstra

Die kürzesten Wege vom Startknoten 0 zu allen anderen Knoten werden wie folgt ermittelt:

1. Starte mit B = {0} und l(0)=0.
2. Setze R = Menge aller Nachbarn von B.
3. Für jede Kante $v_i = [b_i,r_j]$, die einen Knoten $b_i \in B$ mit einem Knoten $r_j \in R$ verbindet, bilde man
   $f(v_i) = l(b_i) + w(v_i)$
4. Wähle dasjenige v=[b,r] mit minimalem $f(v_i)$ aus und füge dessen r zu B hinzu. Setze
   l(r) = f(v) und W(r) = b (Vorgänger auf Weg nach r)
5. Solange B noch nicht alle Knoten enthält: Weiter bei 2.

Übung: Ermitteln Sie den Kürzeste-Wege-Baum **vom Startknoten 6** mit dem Dijkstra-Verfahren.

Eine schöne Veranschaulichung des Dijkstra-Verfahrens findet sich auch im "Algorithmus der Woche" aus dem **Informatikjahr 2006**, s.
www-i1.informatik.rwth-aachen.de/~algorithmus/, 7. Algorithmus
(oder auch www.informatikjahr.de/index.php?id=193, schönere Bilder zu jedem Algo, aber die letzten beiden Wochen fehlen)

In Vorlesung besprechen wir ein berühmtes Graphenproblem, für das es keine einfache Lösung gibt: das **TSP** [Aig99] [Brill, S. 240].

Beispiel *Rheinlandproblem*:

	Aachen	Bonn	Düsseldorf	Frankfurt	Köln	Wuppertal
Aachen		91	80	259	70	121
Bonn	91		77	175	27	84
Düsseldorf	80	77		232	47	29
Frankfurt	259	175	232		189	236
Köln	70	27	47	189		55
Wuppertal	121	84	29	236	55	

Welches ist die kürzeste Tour, die genau einmal durch jede Stadt geht?

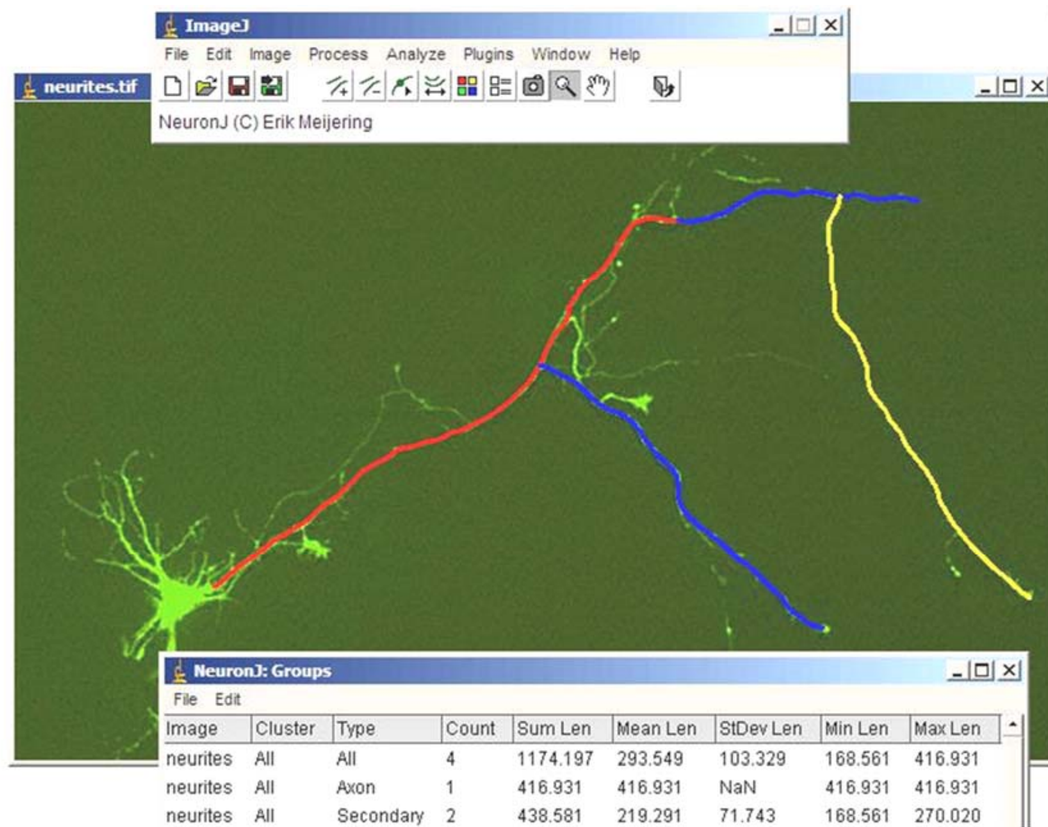
Für das TSP ist bis heute kein effizientes Verfahren zur exakten Lösung bekannt; aber effiziente Verfahren, die gute Näherungslösung ermitteln, liegen vor.

9.4.3. Where to go from here

Graphen finden in vielen Gebieten der Informatik ihre Anwendung (s. Einleitung). Eine interessante Anwendung ist auch in der Bildverarbeitung der sog. **LiveWire**-Algorithmus.

Der Biologe muss in Mikroskopie-Schnittbildern von Neuronen die Länge und Zahl von Dendriten vermessen.

Hierzu liefert die Software ImageJ (http://rsb.info.nih.gov/ij/) und dazu das ImageJ-PlugIn **NeuronJ** einen Beitrag (http://imagescience.bigr.nl/meijering/software/neuronj/):



Die Pixel des Bildes stellen einen Gitter-Graphen dar [Bild zeichnen], die Kanten werden gemäß der Bildinformation bewertet. Dann sucht ein Dijkstra-artiger Algorithmus namens **LiveWire** den kürzesten Weg. Der Algorithmus heisst so, weil er "real time"-schnell ist [vorführen] und weil er an ein auf dem Boden liegendes, tanzendes Hochspannungskabel (engl. "live wire") erinnert.