Mathematische Grundlagen der Informatik

Graphentheorie

W. Gansterer, K. Schindlerová

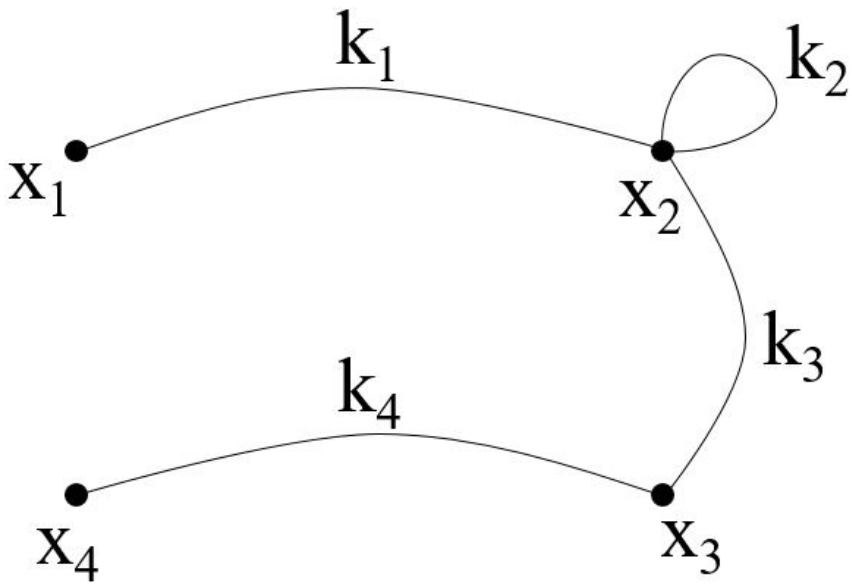
20.05.2020

Graphentheorie

Ungerichtete Graphen

Definition

Ein (ungerichteter) Graph G besteht aus einer Menge $V = V(G)$, den Knoten von G, und aus einer Menge $E = E(G)$ von ungeordneten Paaren k=[x,y] mit $x, y \in V$, den Kanten von G. Man schreibt $G = (V, E)$.



$$V = \{x_1, x_2, x_3, x_4\}$$
$$E = \{k_1, k_2, k_3, k_4\}$$
$$k_1 = [x_1, x_2]$$
$$k_2 = [x_2, x_2]$$
$$k_3 = [x_2, x_3]$$
$$k_4 = [x_3, x_4]$$

Grad

Definition

Ist x ein Knoten des Graphen G, so heißt die Anzahl der mit x inzidenten Kanten Grad von x. Dabei zählen Schlingen doppelt. Der Grad von x wird mit $d(x)$ bezeichnet.

Satz

In jedem Graphen $G = (V, E)$ gilt $\sum d(x) = 2 \cdot |E|$ ... Summe der Grade ist die doppelte Anzahl der Kanten im Graphen, sie ist also immer gerade.

Beweis: Jeder Endpunkt einer Kante liefert genau den Beitrag 1 zum Grad dieses Punktes. Jede Kante hat genau 2 Endpunkte $\Rightarrow$ sie liefert zu der linken Seite der Gleichung genau zweimal den Betrag 1.

Folgerung: In jedem Graphen ist die Anzahl der Knoten mit ungeradem Grad gerade.

Beachten Sie:

Der Grad des Knotens x_n entspricht der Summe der Einträge der n-ten Zeile (bzw. Spalte) der dazugehörigen Adjazenzmatrix.

(*Achtung:* Schlingen doppelt zählen!)

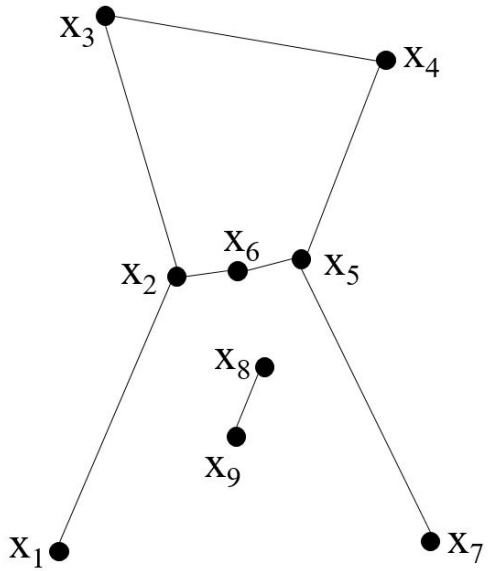
Wege in Graphen

- Eine Menge von Kanten, die den Knoten x_1 mit x_n verbindet, heißt **Kantenfolge** von x_1 nach x_n
- Eine Kantenfolge heißt **offen**, wenn $x_1 \neq x_n$, und **geschlossen**, wenn $x_1 = x_n$

- Ein **Weg** von x nach y ist eine offene Kantenfolge, in der alle Knoten verschieden sind
- Ein **Kreis** ist eine geschlossene Kantenfolge, in der bis auf Anfangs- und Endknoten alle Knoten verschieden sind

- Der Knoten y heißt **erreichbar** vom Knoten x, wenn es einen Weg von x nach y gibt.
- Ein (ungerichteter) Graph heißt **zusammenhängend**, wenn jeder Knoten von jedem erreichbar ist.

Beispiele



- $x_1x_2x_3x_4x_5x_6x_2x_1$ ist eine geschlossene Kantenfolge
- $x_1x_2x_6x_5x_7$ ist ein Weg
- $x_2x_3x_4x_5x_6x_2$ ist ein Kreis
- Der Graph ist nicht zusammenhängend, die Teilgraphen $x_1, x_2, ..., x_7$ und x_8, x_9 sind die **Zusammenhangskomponenten** des Graphen

Länge und Gewicht

Definition

Die Anzahl der Kanten einer Kantenfolge oder eines Weges heiße **Länge** der Kantenfolge bzw. des Weges.

Definition

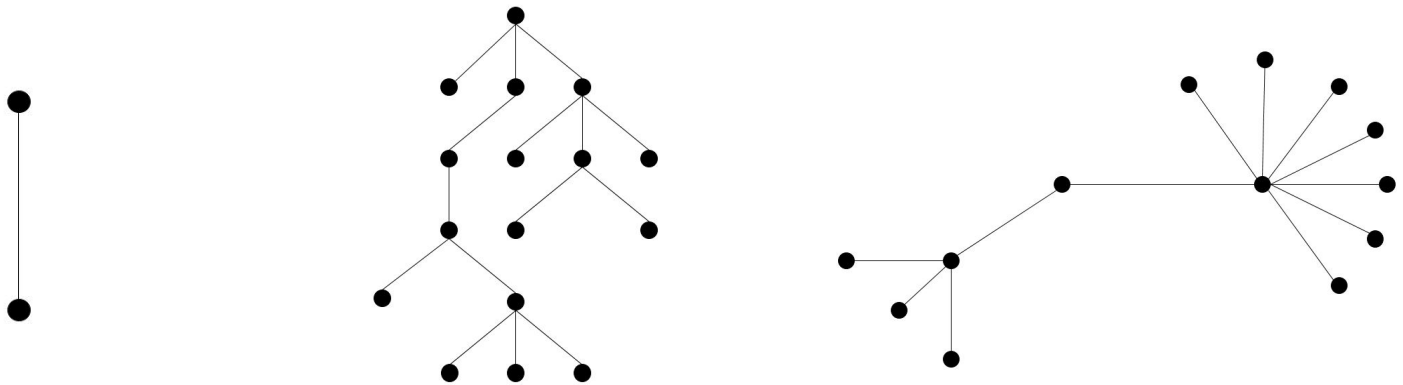
Ein Graph heißt **bewertet/gewichtet**, wenn jeder Kante $[x, y]$ ein **Gewicht** $w(x, y) \in \mathbb{R}$ zugeordnet ist. In bewerteten Graphen ist die *Länge eines Weges* von einem Knoten zu einem anderen die Summe der Gewichte aller Kanten des Weges.

Bäume

Bäume

Definition

Ein Graph, in dem je zwei Knoten durch genau einen Weg verbunden sind, heißt **Baum**. Ein Baum ist also ein zusammenhängender Graph ohne Kreise.



Charakterisierung Baum

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

a) G ist ein Baum (je zwei Knoten durch genau einen Weg verbunden).

b) G ist ein zusammenhängender Graph ohne Kreise.

c) G ist zusammenhängend, aber entfernt man eine beliebige Kante, so zerfällt G in zwei Zusammenhangskomponenten.

d) G ist zusammenhängend und $n = m + 1$.
(G hat einen Knoten mehr als Kanten)

Charakterisierung Baum

Beweis.

- a) $\Rightarrow$ b): G ist natürlich zusammenhängend. Hätte G einen Kreis, so gäbe es zwei verschiedene Wege zwischen den Knoten dieses Kreises.
- b) $\Rightarrow$ c): Nehmen wir eine Kante $E = [x_1, x_2]$ weg, so erhalten wir den Graphen G', in dem x_1 und x_2 nicht mehr verbunden sind, G' ist also nicht mehr zusammenhängend (sonst hätte es in G einen Kreis gegeben).
  Sei y irgendein Knoten. Wenn y in G' nicht mehr mit x_1 verbunden ist, so enthielt der ursprüngliche Weg von y nach x_1 die Kante E und damit auch x_2. Der Weg nach x_2 existiert also noch, y ist mit x_2 verbunden. G' zerfällt also in zwei Komponenten: Die Knoten einer Komponente sind alle mit x_1 verbunden, die Knoten der anderen Komponente alle mit x_2.

Charakterisierung Baum

Beweis.

- c) $\Rightarrow$ a): Da G zusammenhängend ist, sind je zwei Knoten durch genau einen Weg verbunden. Gäbe es zwei verschiedene Wege von x_1 nach x_2, so könnte man aus einem dieser Wege eine Kante wegnehmen, ohne den Zusammenhang zu zerstören, was im Widerspruch zur Annahme c) steht.

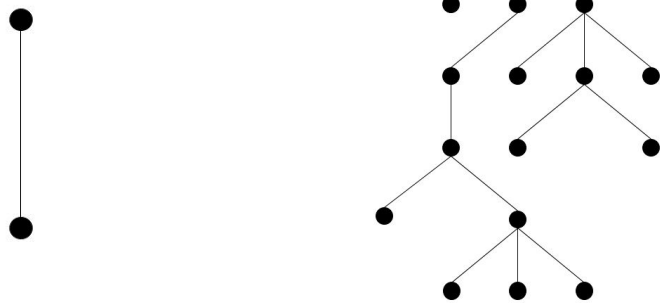
Charakterisierung Baum

Beweis.

- Skizze für d) $\Leftrightarrow$ a), b), c):
  - Vorbereitung: Nach Wegnahme einer Kante aus einem Baum bilden die entstehenden Zusammenhangskomponenten wieder Bäume: Sind z. B. y und z Knoten in einer solchen Komponente, so kann es nicht mehrere verschiedene Wege von y nach z geben, da diese auch im ursprünglichen Graphen G Wege wären.
  - Zeige a) $\Rightarrow$ d) mit vollständiger Induktion nach der Anzahl der Knoten n.
  - Zeige d) $\Rightarrow$ a) mit vollständiger Induktion nach der Anzahl der Knoten n (unter Beachtung von $\sum_{x \in V} d(x) = 2 \cdot |E|$).

Charakterisierung Baum

Zeige a) $\Rightarrow$ d) mit vollständiger Induktion nach der Anzahl der Knoten n.
Der Induktionsanfang ($n = 1$ und $n = 2$) lässt sich aus dem Bild ablesen.
Es soll nun für alle Bäume mit weniger als n + 1 Knoten die Behauptung erfüllt sein.
Sei $G = (V, E)$ $|V| = n + 1$. Nehmen wir eine Kante weg, so erhalten wir 2 Teilbäume mit Knotenzahl n_1 bzw. n_2, wobei $n1 + n2 = n + 1$ gilt.
Für die Teilbäume ist die Kantenzahl nach Voraussetzung $n_1 - 1$ bzw. $n_2 - 1$. Damit ergibt sich für die Gesamtzahl der Kanten von G : $n_1 - 1 + n_2 - 1 + 1 = n$.



Charakterisierung Baum

Zeige d) $\Rightarrow$ a) auch mit vollständiger Induktion nach der Anzahl der Knoten n.
Induktionsanfang - das Bild.
Sei jetzt ein Graph $G = (V, E)$, $|V| = n + 1, |E| = n$. Der Grad jedes Knotens ist mindestens 1, da G zusammenhängend ist.
Dann muss es Knoten vom Grad 1 geben, also Knoten, die mit genau einer Kante inzidieren, sondern wäre für alle Knoten $d(x) \geq 2$, so würde unter Beachtung von $\sum_{x \in V} d(x) = 2 \cdot |E|$ folgen:

$$2(n+1) \leq \sum d(x) = 2|E| = 2n.$$

Charakterisierung Baum

Entfernen wir jetzt einen solchen Knoten x vom Grad 1 mit seiner Kante aus G, so entsteht ein kleinerer Graph, der wieder genau einen Knoten mehr hat als Kanten.
Dieser ist also nach Induktionsannahme ein Baum.
Nimmt man zu diesem Baum wieder x mit seiner Kante hinzu, so ist auch x mit jedem anderen Knoten des Graphen durch genau einen Weg verbunden und damit ist G ein Baum.

Charakterisierung Baum

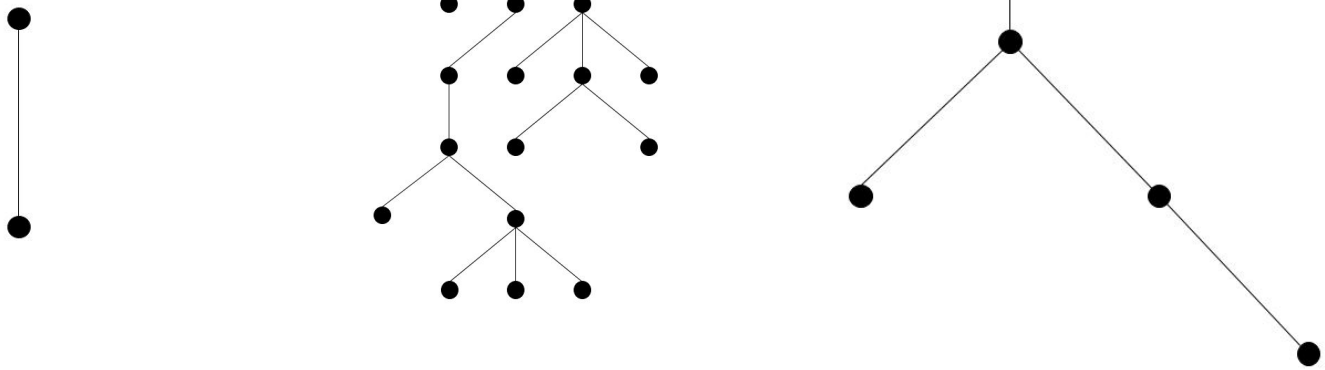
Zusammenfassung:
Für einen Graphen G mit n Knoten und m Kanten sind die folgenden Aussagen äquivalent:

1. G ist ein Baum (je zwei Knoten durch genau einen Weg verbunden).
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.

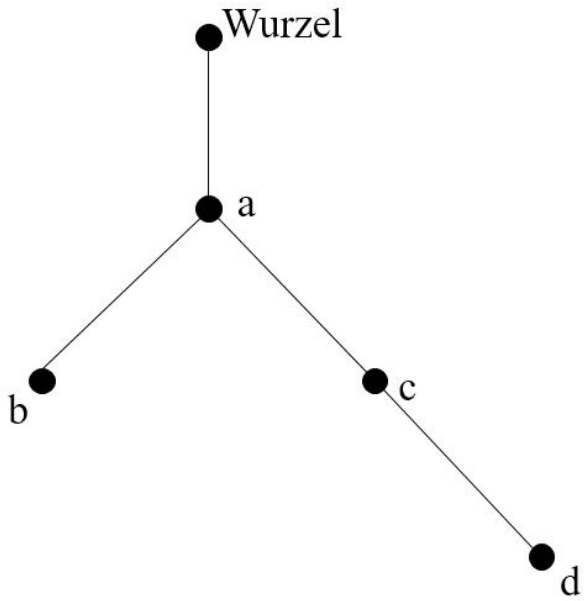
Die Baumwurzel wird in der Regel oben gezeichnet, alle Kanten weisen nach unten.



Wurzelbaum – Definitionen

- Sind x und y durch 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.

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

Wurzelbaum

Beachten Sie:

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

$\Rightarrow$ (Wurzel)Bäume eignen sich hervorragend für rekursive Bearbeitung!

- Teilbäume haben die gleichen Eigenschaften wie der ursprüngliche Baum, werden aber immer kleiner
- Viele Baumalgorithmen sind daher rekursiv aufgebaut!

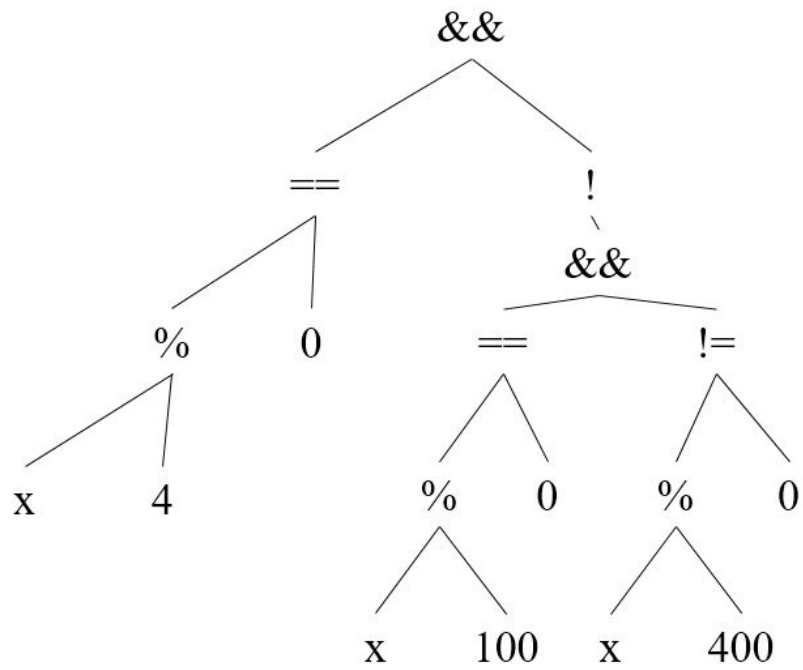
Anwendung: Compiler

Syntaxanalyse: Der Compiler erzeugt aus dem zu verarbeitenden Programmcode 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.

Beispiel: Syntaxbaum für

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



Anwendung: Compiler

Nach dem Aufbau des Syntaxbaums kann der Compiler den Ausdruck *rekursiv* auswerten:

- 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 höchstens n Söhne, so heißt er **n-ärer Wurzelbaum**.
- Hat jeder Knoten *genau* n oder 0 Söhne, so heißt er **regulärer n-ärer Wurzelbaum**.
  - Spezialfall $n = 2 \rightarrow$ binäre (reguläre) Wurzelbäume, die Söhne werden *linke* und *rechte* Söhne genannt.

Eigenschaften binärer regulärer Wurzelbäume

Jeder binäre reguläre Wurzelbaum B hat folgende Eigenschaften:

a) Falls B mehr als einen Knoten hat, dann gibt es genau einen Knoten vom Grad 2, und alle anderen Knoten haben Grad 1 oder 3.

b) Ist x ein Knoten von B, so bildet x mit all seinen Nachkommen wieder einer binären regulären Wurzelbaum.

c) Die Anzahl der Knoten ist ungerade.

d) Hat B n Knoten, so hat er $(n+1)/2$ Blätter und $(n-1)/2$ innere Knoten. Es gibt also genau ein Blatt mehr als innere Knoten.

e) In einem binären Wurzelbaum der Höhe H mit n Knoten gilt:

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

Eigenschaften binärer regulärer Wurzelbäume

- Beweisen wir Aussage a):
  - Die Wurzel hat genau zwei Söhne, also Grad 2, jeder andere Knoten ist ein Sohn und hat entweder zwei Söhne (Grad 3) oder keine Nachkommen (Grad 1).
- Beweisen wir Aussage b):
  - Die charakterisierenden Eigenschaften des binären regulären Baumes bleiben natürlich in allen Teilbäumen erhalten.

Eigenschaften binärer regulärer Wurzelbäume

Zum Beweis von Aussage c): Mittels Induktion über die Anzahl der Knoten!

- Ein minimaler binärer regulärer Wurzelbaum ist ein einzelner Knoten, für diesen stimmt die Aussage.
- Die Anzahl der Knoten des Baumes B ist 1 + Anzahl der Knoten des linken Teilbaums + Anzahl der Knoten des rechten Teilbaums $\rightarrow$ also ungerade, da die beiden kleineren Teilbäume *laut Induktionsannahme* ungerade Knotenzahl haben.

Eigenschaften binärer regulärer Wurzelbäume

Zum Beweis von Aussage d): Sei p die Anzahl der Blätter von B.

- Wenn B n Knoten hat, so hat B n–1 Kanten. [*gilt in jedem Baum!*]
- Laut Aussage 1 und da in jedem Graphen gilt $\sum d(x) = 2 \cdot |E|$ [*vgl. früher!*]
- $\sum d(x) = p \cdot 1 + (n - p - 1) \cdot 3 + 1 \cdot 2 \quad = \quad 2|E| = 2(n-1)$
- Anzahl der Blätter: Nach p auflösen,
  $\sum d(x) = -2p + 3(n-1) + 2 = 2(n-1)$
  $\Rightarrow -2p + (n-1) = -2 \Rightarrow p = (n+1)/2$
- Anzahl der inneren Knoten:

$$n - p = n - (n+1)/2 = (n-1)/2 = p - 1.$$

Eigenschaften binärer regulärer Wurzelbäume

Zum Beweis von Aussage e) [$H \geq log_2(n+1)$]:

- Mittels Induktion kann man leicht zeigen, dass sich auf Niveau k des Baumes maximal 2^{k-1} Knoten befinden.
- Auf dem untersten Niveau H befinden sich also maximal 2^{H-1} Knoten, die dann alle Blätter sind.
- Addiert man dazu die inneren Knoten, so erhält man [*beachten Sie Aussage d)!*]
  $n \leq 2^{H-1} + (2^{H-1}-1)$.
- Daraus ergibt sich also $n \leq 2 \cdot 2^{H-1}-1 = 2^H-1$ und damit

$$\log_2(n+1) \leq H.$$

Suchbäume

Aussage e) von vorher 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

- Datensätze werden durch Schlüssel identifiziert.
- Jedem Datensatz wird ein (eindeutiger) Schlüssel zugeordnet.
- Jedem Knoten wird ein Datensatz zugeordnet.

- *Eintragen von Schlüssel und Daten 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.

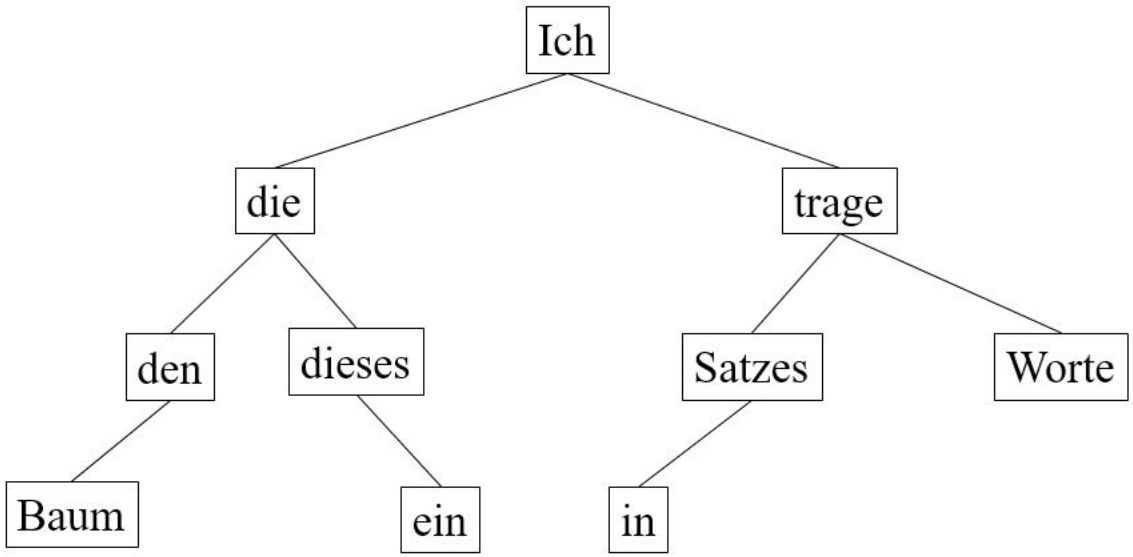
$s < w \Rightarrow$ Trage Schlüssel s im linken Teilbaum der Wurzel ein.

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

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 umso effizienter, je weniger sich die Höhen linker und rechter Teilbäume unterscheiden $\Rightarrow$ *höhenbalancierte Bäume*

Suche nach Daten mit Schlüssel s

Auch das Suchen eines Datensatzes in einem solchen Baum erfolgt rekursiv:
Den Satz, der zu einem gegebenen Schlüssel s gehört, finden wir mit dem folgenden Algorithmus:
Die Suche nach dem Schlüssel s beginnend bei dem Knoten x
(x hat den Schlüssel s_x)

$$s = s_x \Rightarrow \text{Daten gefunden.}$$
$$s < s_x \Rightarrow \text{Setze die Suche beim linken Sohn von } x \text{ fort.}$$
$$s > s_x \Rightarrow \text{Setze die Suche beim rechten Sohn von } x \text{ fort.}$$

Damit wird jedes eingetragene Datum gefunden.
Um den Fall abzudecken, dass nach einem nicht vorhandenen Schlüssel gesucht wird, muss der Algorithmus etwas erweitert werden.

Suchbäume

Alle in dem Baum enthaltenen Daten können Sie alphabetisch ausgeben mit der Vorschrift:
Gebe den Baum mit Wurzel x alphabetisch aus:

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

Traversierungsverfahren – Durchlaufen von Suchbäumen

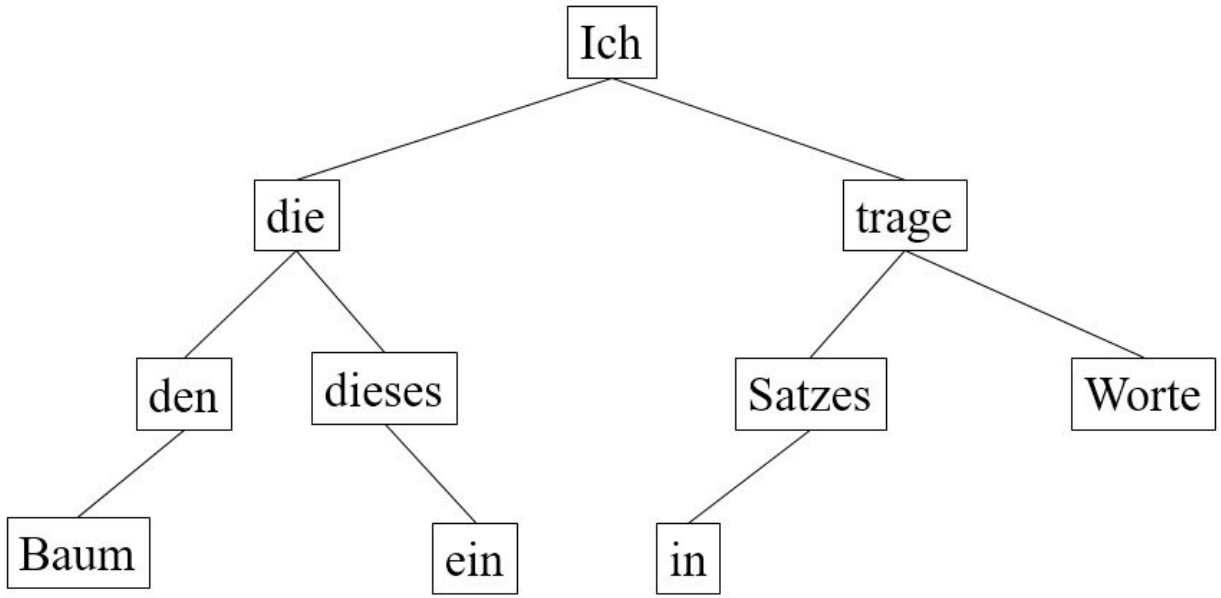
Dies ist ein Algorithmus, der systematisch alle Knoten eines Graphen besucht. Solche Traversierungsverfahren sind grundlegender Bestandteil vieler Graphenalgorithmen.
Der vorgestellte Algorithmus beschreibt den Inorder-Durchlauf eines binären Baumes.
Inorder-Durchlauf: Die Daten des Baums im Beispiel mit Wurzel x werden in alphabetischer Reihenfolge ausgegeben.

Beispiel 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 ein dieses die in Satzes Worte trage Ich“
In diesen wird lediglich die Wurzel nicht zwischen den Teilbäumen, sondern vor beziehungsweise nach den Teilbäumen besucht.



Anwendung binäre Bäume: Huffman-Code

Ein wichtige Algorithmus zur Datenkompression

Grundidee: Kapazität bei Datenübertragung einsparen

- Häufig vorkommende Zeichen durch kurze Codeworte ersetzen
- Selten vorkommende Zeichen durch längere Codeworte ersetzen
- Beispiel: Morsealphabet

Ziel: Minimierung von benötigtem Speicherplatz durch *variable Codewortlänge*

- Häufige Zeichen möglichst kurz codieren
- Im Gegensatz dazu: ASCII-Code – für jedes Zeichen benötigt man genau ein Byte, für das häufige „E“ ebenso wie für das seltene „X“.

Präfix Codes

Problem bei Codes variabler Länge: Trennung von Codewörtern („wo hört eines auf, wo beginnt das nächste?“)

Lösungen:

- Trennzeichen (z.B. Morsecode), z.B.

$$E = .\ T = -\ A = .-\ \ , H = ...\ X = -..-$$

Bei solchen Codes variabler Länge gibt es jedoch ein Decodierungsproblem: Wie wird zum Beispiel beim Empfänger der Code „.-“ interpretiert? Als „A“ oder als „ET“?

Präfix Codes

Kann man einen Code so aufbauen, dass man auf das Trennzeichen zwischen zwei Codeworten verzichten kann?

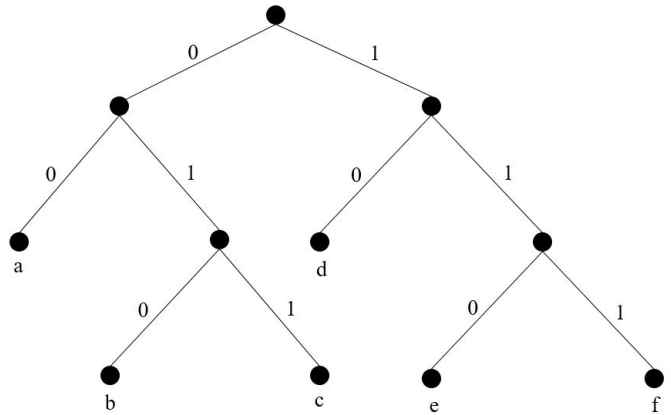
Lösungen:

- *Präfix-Codes – kein Codewort ist Anfangsteil eines anderen Codeworts*
  - Bsp: Telefonnummernsystem - keine Telefonnummer beginnt mit “110”
    Das Ende eines Codewortes wird ohne weiteres Trennzeichen erkannt: Wenn Sie 110 wählen, weiß die Vermittlung, dass die Telefonnummer zu Ende ist, weil keine andere den gleichen Anfang hat.
    Das Morsealphabet ist kein Präfixcode: Der Code von „E“ ist Anfangsteil des Codes von „A“.

Präfix-Codes können mithilfe von Wurzelbäumen konstruiert werden.

Konstruktion von Präfix-Codes mittels Wurzelbäumen

- Codeworte bestehen nur aus „0“ und „1“ = Kantenfolgen in einem binären regulären Wurzelbaum
  - Kanten nach links: “0”, Kanten nach rechts: “1”
- Blätter enthalten die Zeichen, die kodiert werden sollen



$\Rightarrow$ Kein Codewort ist Anfangsteil eines anderen (sonst müsste das dazugehörige Zeichen auf dem Weg zu diesem anderen Wort liegen). Wir haben aber nur Blättern kodiert. Man kann ein Bitstrom aus diesen Codeworten eindeutig dekodieren, ohne dass man ein Pausezeichen verwendet.

Konstruktion von Präfix-Codes mittels Wurzelbäumen

- Beispiel: „0100110010“ eindeutig auflösbar in „010 011 00 10“
  b e a d
- Umgekehrt lässt sich jeder Präfix-Code durch einen solchen Baum darstellen.

Konstruktion:
Beginne bei der Wurzel und zeichne für jedes Codewort die Kantenfolge auf, die jeweils für 0 zum linken Sohn, für 1 zum rechten Sohn führt.
Wegen der Präfix-Eigenschaft sind die Codeworte genau die Kantenfolgen, die in den Blätter des entstehenden Baums enden.

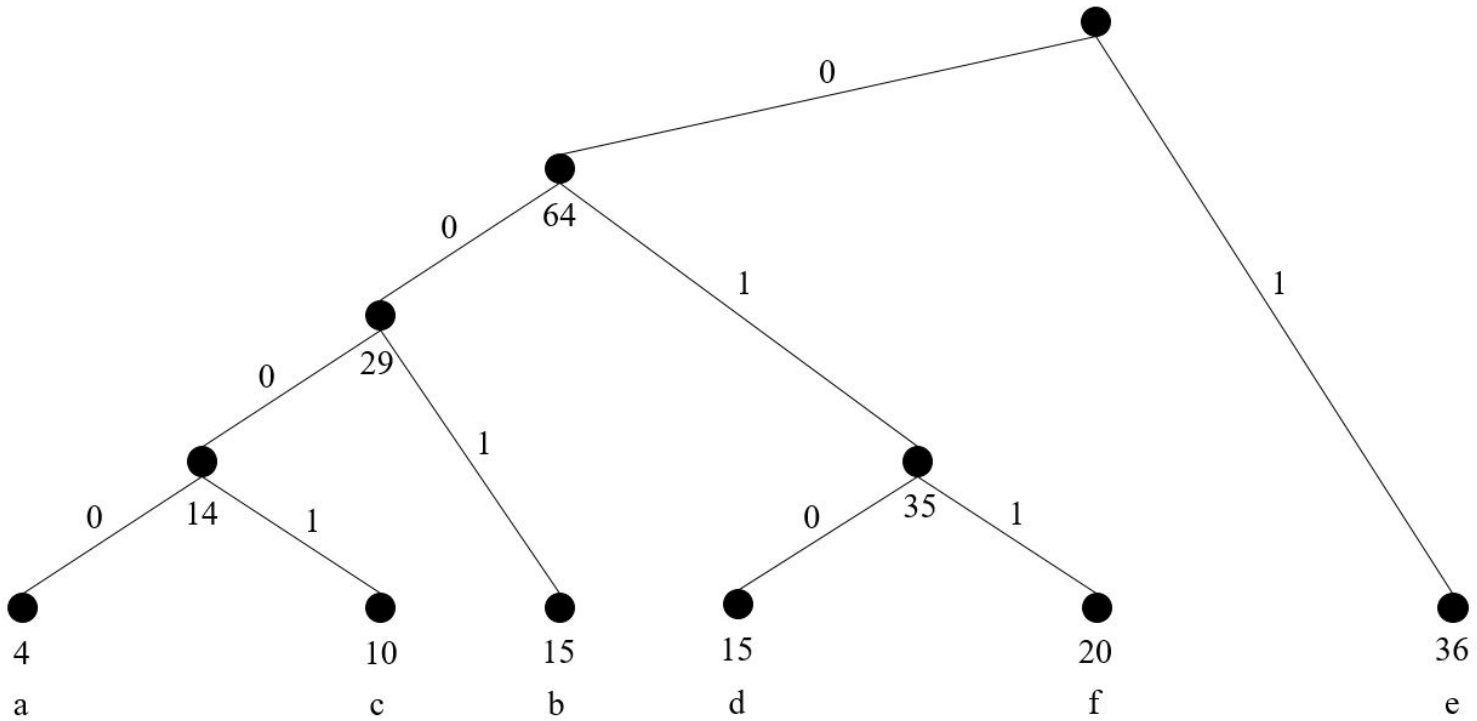
Optimaler Präfix-Code: Huffman-Code

Häufig vorkommende Zeichen kurze Codewörter, seltene Zeichen längere Codewörter $\Rightarrow$ Huffman-Algorithmus (produziert optimalen Präfix-Code)

- **Gegeben:** Quellalphabet ($\{a, b, c, d, e, f\}$), Häufigkeit des Auftretens der Zeichen, z. B.
  $a : 4\%$ $b : 15\%$ $c : 10\%$ $d : 15\%$ $e : 36\%$ $f : 20\%$

- **Algorithmus:**
  1. Die Zeichen des Alphabets werden die Blätter eines binären Baumes. Die Blätter enthalten Häufigkeiten der Zeichen (aufsteigend geordnet).
  2. 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.
  3. Wiederhole Schritt 2 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, bei der man recht gut die Wahrscheinlichkeiten für die Anzahl aufeinanderfolgender weißer oder schwarzer Bildpunkte kennt,
  - als Teil des jpeg-Algorithmus zur Bildkomprimierung

Anwendung: ZIP-Codierung

Präfix-Codes codieren Zeichen konstanter Länge in unterschiedlich lange Codeworte.

- Ein alternativer Kompressionsansatz sucht im Quelltext häufig vorkommende Worte, auch unterschiedlicher Länge, und übersetzt diese in Codeworte konstanter Länge.
- ZIP-Codierung verwendet en solches Verfahren: Während der Codierung wird ein Wörterbuch häufig vorkommender Zeichenketten erstellt und – ebenfalls unter Verwendung von Bäumen – diesen Zeichenketten ein Codewort fester Länge zugeordnet.