Bäume*

Hans Jürgen Ohlbach

23. März 2018

Empfohlene Vorkenntnisse: Graphen

Inhaltsverzeichnis

*Dieser Text ist Teil einer Sammlung von Miniskripten zur Einführung in die Informatik. Er ist in erster Linien für Nichtinformatiker gedacht, kann aber natürlich auch als erste Einführung für Informatiker nützlich sein.

1 Übersicht

Bäume in der Informatik sind eine spezielle Art von Graphen. Solche Strukturen kommen in der Informatik fast noch häufiger vor als Graphen. Es gibt sie in sehr unterschiedlichen Varianten mit unterschiedlichen Eigenschaften und Algorithmen. Daher sind den Bäumen eine ganze Reihe von Miniskripten gewidmet. Wir geben zunächst einen groben Überblick über die verschiedenen Typen.

Gerichtete Bäume und ungerichtete Bäume
Gerichtete Bäume haben eine Wurzelknoten und wachsen dann hin zu den Blattknoten. Ungerichtete Bäume sind im wesentlichen zyklenfreie ungerichtete Graphen. Ein Beispiel sind die Spannbäume (siehe Abschnitt 3 und Kap. Spannbäume in GewichteteGraphen.pdf).

Geordnete und ungeordnete Bäume
Bäume in der Informatik machen nur Sinn, wenn die Knoten Zusatzinformationen tragen, sog. *Knotenlabel* oder *Schlüssel*, die von der Anwendung her kommen. Bei der Art von Zusatzinformation gibt es jedoch zwei grundlegende Unterschiede, die sich ganz wesentlich auf die Algorithmen für die Bäume auswirken:

- Die Knotenlabel haben keine direkte Beziehung zueinander. Das ist insbesondere der Fall wenn der Baum dynamisch generiert wird, z.B. bei nichtdeterministischen Algorithmen. In diesem Fall stellen die Knoten Verzweigungspunkte dar, von welchen aus es verschiedene Alternativen gibt. Im Miniskipt *Baumsuche* werden dazu die Standardsuchverfahren *Breitensuche*, *Tiefensuche*, und *Iterative Deepening* eingeführt, mit denen man die verschiedenen Alternativen nacheinander absuchen kann.
- Auf den Knotenlabel gibt es eine Ordnung, i.A. sogar eine *Totalordnung*. Ordnet man die Knoten im Baum entsprechend der Ordnung an, dann kann man den Baum extrem effizient für die Suche nach bestimmten Labeln verwenden.

Bäume, deren Knotenlabel geordnet sind, eignen sich ganz besonders, um Mengen von Objekten in einer sortierten Reihenfolge zu halten, sowie die sortierte Reihenfolge zu nutzen, um nach einem bestimmten Knotenlabel zu suchen. Daher werden sie auch *Suchbäume* genannt. Im Gegensatz zu Arrays, deren Einträge man natürlich auch sortieren kann, die aber eine feste Größe haben, eignen sich Bäume für Anwendungen, wo sehr oft neue Objekte eingetragen werden, und alte Objekte gelöscht werden. Da das eine sehr häufig vorkommende Aufgabe ist, wurden im Laufe der letzten Jahrzehnte eine Reihe unterschiedlicher Baumtypen entwickelt, die für unterschiedliche Anwendungen optimiert sind:

Heaps sind Bäume, wo auf effiziente Weise der größte Knotenlabel an die Wurzel befördert werden kann. Die restlichen Knotenlabel müssen dann nicht sortiert sein. Anwendungen davon sind z.B. Prioritätsschlangen, wo man an der Spitze der Schlange das größte Element haben will. Der Rest ist zunächst irrelevant. Das Sortierverfahren *Heapsort* basiert auf diesen Heaps (siehe Miniskript Sortieren.pdf).

Heaps werden im Miniskript Heaps.pdf besprochen.

AVL-Bäume sind *Binärbäume* (siehe Abschnitt 2.1), d.h. die Knoten haben maximal zwei Kindknoten, wo alle Pfade bis zu den Blattknoten ungefähr gleich lang sind (± 1). Damit kann man jeden Knoten in logarithmischer Zeit finden.

AVL-Bäume werden im Miniskript AVLBaeume.pdf besprochen.

Rot-Schwarz-Bäume sind ebenfalls Binärbäume, die ein etwas anderes Kriterium erfüllen, und bei denen man jeden Knoten aber auch in logarithmischer Zeit findet. Bei Vergleichsstudien haben AVL-Bäume leicht besser abgeschnitten als Rot-Schwarz-Bäume. Daher werden Rot-Schwarz-Bäume nicht weiter besprochen.

B-Bäume sind ebenfalls Suchbäume, bei denen man auf sehr schnelle Weise Knoten finden kann. Sie sind aber keine Binärbäume, sondern jeder Knoten kann mehr als zwei Kindknoten haben. Sie werden insbesondere für Datenbanken eingesetzt. Dabei richtet sich die Anzahl der Kindknoten eines Knotens nach der Speicherstruktur der Datenbank. Eine spezielle Art von B-Bäumen sind die 2-3-4 Bäume, die 2 - 4 Kindknoten pro Knoten haben.

B-Bäume werden im Miniskript BBaeume.pdf besprochen.

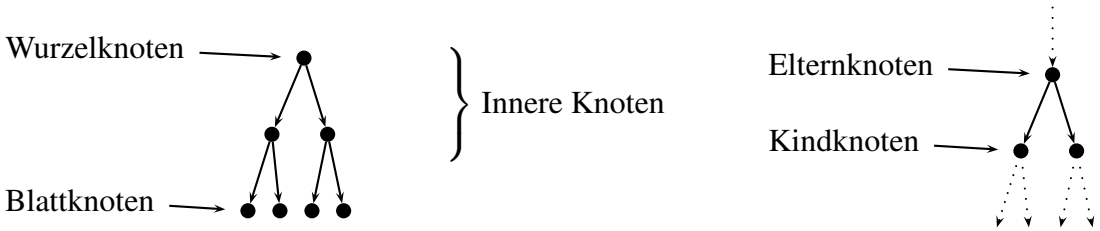
R-Bäume unterstützen die Suche nach Punkten und ausgedehnten Objekten in 2- und mehrdimensionalen Koordinatensystemen. Sie werden im Miniskript RBaeume.pdf besprochen.

Bei den meisten Baumtypen ist es ganz wichtig, dass sie *balanciert* sind. Das heißt im wesentlichen, dass die Unterschiede der Pfadlängen von der Wurzel bis zu den Blattknoten möglichst gleich, und als Konsequenz davon auch möglichst gering sind. Nur dann ist die Suche nach einem bestimmten Knoten effizient. Das bedeutet aber, dass das Einfügen und Löschen eines Knotens die Balanciertheit erhalten muss, was oft einiges an Umbauten im Baum erfordert.

Im vorliegenden Miniskript werden nur die grundlegenden Begriffe eingeführt, als Basis für die anderen Miniskripte, in denen konkrete Algorithmen mit Bäumen vorgestellt werden.

2 Gerichtete Bäume (engl. rooted trees)

Ein gerichteter Baum startet mit einem *Wurzelknoten* (engl. root node). Jeder Knoten ist entweder ein *Blattknoten* (engl. leaf node), oder er hat eine Reihe von *Kindknoten* (*Unterknoten*). Jeder Knoten außer dem dem Wurzelknoten hat *genau einen Elternknoten*. Graphisch kann das so aussehen:



In der Informatik zeichnet man Bäume nicht so wie in der Natur, Wurzel unten und Blätter oben, sondern umgekehrt, Wurzel oben und Blätter unten.

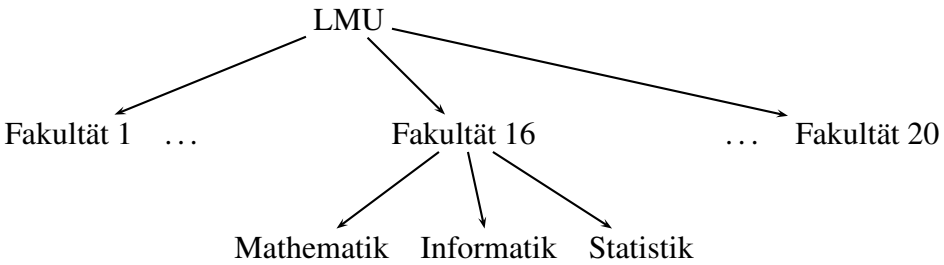
Mehrere von solchen Bäumen nennt man einen *Wald* (engl. forest).

Daten, die man auf diese Weise baumartig modellieren kann gibt es enorm viele.

Beispiele sind:

- Die Vererbungshierarchie in der Klassenstruktur von Java ist baumförmig. Das liegt an der sog. Monovererbung, d.h. jede Klasse hat maximal eine Oberklasse. (In der Programmiersprache C++ gibt es Multivererbung. Da bildet die Vererbungshierarchie einen DAG (Directed Acyclic Graph)).

- Viele Organigramme sind baumförmig strukturiert.
  Beispiel:

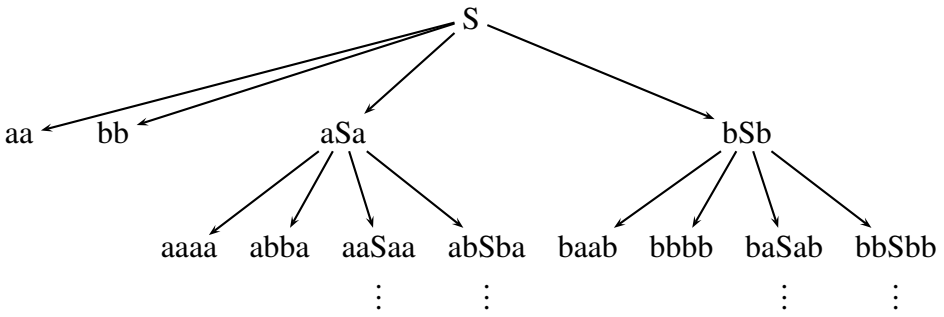


- Stammbäume von Personen sind i.A. auch baumförmig strukturiert, zumindest bis zu der Ebene, ab der unterschiedliche Vorfahren wieder gemeinsame Vorfahren haben. Dann wird aus der Struktur ein DAG.

- Ableitungsbäume von Grammatiken in Formalen Sprachen sind baumförmig organisiert. Als Beispiel betrachten wir die Grammatik für Palyndrome über den Buchstaben a und b.

  $$S \rightarrow aSa \mid bSb \mid aa \mid bb$$

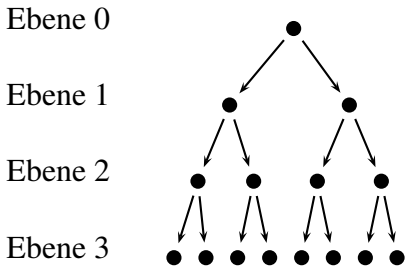
  Diese Grammatik erzeugt einen speziellen Baum, der die Ableitungsmöglichkeiten darstellt:



  Der Baum ist unendlich, aber durch die Grammatik auf endliche Weise beschrieben. Solche Bäume werden nicht explizit repräsentiert, sondern die Knoten und Kanten werden durch entsprechende Funktionen bei Bedarf erzeugt.

2.1 Binärbäume

Von ganz besonderen Interesse sind *Binärbäume*, wo jeder Knoten maximal 2 Kindknoten hat.



Ein binärer Baum

Es müssen dabei nicht alle inneren Knoten genau 2 Kindknoten haben. Aber es dürfen nicht mehr als 2 Kindknoten sein.

Diese Bäume sind deswegen besonders interessant, weil sich damit sehr effizient *binäre Suche* implementieren lässt, und zwar in einem Umfeld, wo man häufig neue Knoten hinzufügt und wieder löscht. Das geht bei Bäumen deutlich effizienter als bei Arrays oder Listen.

Bäume sind als sog. *Indexstrukturen* ganz wichtige Komponenten in Datenbanksystemen. Sie unterstützen das effiziente Auffinden von Daten in großen Datenbanken.

2.2 Eigenschaften von gerichteten Bäumen

Ein Pfad in einem Baum ist eine Folge von Knoten vom Wurzelknoten bis zu einem bestimmten Knoten. Da jeder Knoten nur maximal einen Elternknoten hat, ist ein Pfad immer eindeutig.

Die Tiefe eines Knotens in einem Baum ist die Länge des Pfades bis zu diesem Knoten. Knoten mit gleicher Tiefe definieren eine *Ebene* in dem Baum.

Die Tiefe eines Baumes ist die Länge des längsten Pfades in dem Baum.

Der Grad eines Knotens ist die Anzahl seiner Unterknoten.
Bei regulären Bäumen ist der Grad aller Knoten gleich. Hat ein Baum einen Grad k, dann hat jeder Knoten *maximal* k Unterknoten.
Der Baum ist *vollständig* falls er, bis auf die Blattknoten, *genau* k Kindknoten hat.

Maximale Anzahl von Knoten in einem Baum mit Grad k:
Für die Zeitkomplexität vieler Algorithmen, die auf Bäumen arbeiten, ist es wichtig die maximale Anzahl Knoten und die Tiefe des Baumes abzuschätzen.

Bei binären Bäumen ergeben sich folgende Zusammenhänge:

Ebene	Anzahl Knoten
0	1
1	2
2	4
3	8
k	2^k

Die Blattebene k hat also maximal 2^k Blattknoten.
Bis zur Ebene k hat man $2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1$ Knoten.
Ein binärer Baum mit n Blattknoten hat eine Tiefe von $\lceil \log_2(n) \rceil$.
Mit insgesamt n Knoten hat er eine Tiefe von $\lceil \log_2(n) \rceil$.

Für einen Baum mit Grad g ergibt sich dann analog:
Die Blattebene k hat maximal g^k Blattknoten.
Er, hat alle zusammen, maximal $\frac{g^{k+1}-1}{g-1}$ Knoten.[1] Bei n Blattknoten hat er eine Tiefe von $\lceil \log_g(n) \rceil$.

Da sich Logarithmen mit verschiedenen Basen nur durch eine Konstante unterscheiden, spielt für Komplexitätsbetrachtungen der Grad keine Rolle. Die Tiefe des Baumes ist immer in $\mathcal{O}(\log(n))$.

2.3 Operationen auf Bäumen

Eine Reihe von Operationen auf Bäumen haben sich als nützlich und häufig vorkommend herausgestellt.

Einfügen eines neuen Knotens.

Löschen eines Knotens oder eines ganzen Teilbaums.

Zugriff auf den ersten (linkesten) oder letzten (rechtesten) Knoten.
Bei Suchbäumen ist das der Knoten mit dem kleinsten bzw. größten Knotenlabel.

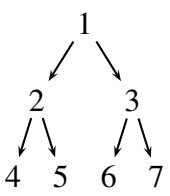
Suche nach einem bestimmten Knotenlabel (Schlüssel).
Bei Suchbäumen macht man das mit jeweils angepasster Version von binärer Suche.
Bei ungeordneten Bäumen muss man den Baum traversieren, bis man den richtigen Knoten findet.

Traversieren der Knoten in bestimmter Reihenfolge.

[1] Der Beweis für interessierte Leser:

$$\begin{aligned} S &= 1 + g + g^2 + \ldots + g^k \\ g \cdot S &= \quad g + g^2 + \ldots + g^k + g \cdot g^k \\ \hline g \cdot S - S &= -1 + 0 + \ldots + 0 + g \cdot g^k \\ (g-1) \cdot S &= g^{k-1} - 1 \\ S &= (g^{k-1} - 1)/(g-1) \end{aligned}$$

Zum Traversieren gibt es verschiedene Möglichkeiten, die an folgendem Baum illustriert werden:



Breadth-First:		1 - 2 - 3 - 4 - 5 - 6 - 7
Depth-First Pre-Order	(K-l-r):	1 - 2 - 4 - 5 - 3 - 6 - 7
Depth-First Post-Order	(l-r-K):	4 - 5 - 2 - 6 - 7 - 3 - 1
Depth-First In-Order	(l-K-r):	4 - 2 - 5 - 1 - 6 - 3 - 7
Depth-First Reverse-In-Order	(r-K-l):	7 - 3 - 6 - 1 - 5 - 2 - 4

Breadth-First (Breitensuche) geht den Baum Ebene für Ebene durch.

Depth-First Pre-Order (Tiefensuche, Hauptreihenfolge) besucht erst den Knoten, dann nacheinander rekursiv die Unterbäume. Bei Binärbäumen ist das die Reihenfolge K-l-r (Knoten, links, rechts).

Depth-First Post-Order (Tiefensuche, Nebenreihenfolge) besucht erst die Unterbäume, und dann den Knoten. Bei Binärbäumen ist das die Reihenfolge l-r-K (links, rechts, Knoten).

Depth-First In-Order Macht nur Sinn bei Binärbäumen und B-Bäumen. Es werden abwechselnd die Unterbäume und die Knotenlabel besucht. Bei Binärbäumen ist das die Reihenfolge l-K-r (links, Knoten, rechts).

Depth-First Reverse-In-Order Macht auch nur Sinn bei Binärbäumen und B-Bäumen. Es werden von rechts nach links abwechselnd die Unterbäume und die Knotenlabel besucht. Bei Binärbäumen ist das die Reihenfolge r-K-l (rechts, Knoten, links).

2.4 Adressierung von Knoten

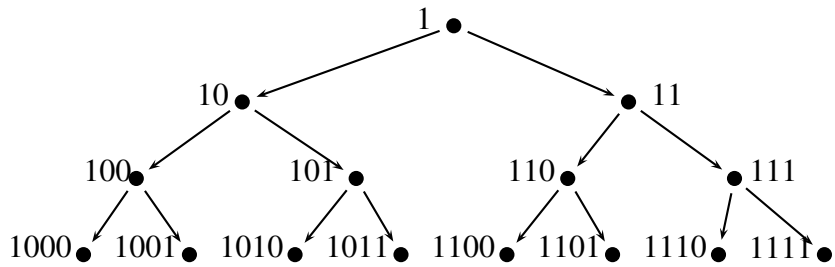
In Bäumen, wo die Kindknoten eines Knotens eine feste Reihenfolge haben, bei Binärbäumen z.B. linker Knoten - rechter Knoten, können die Pfade und Knoten auf einfache Weise als Ziffernfolge adressiert werden:

die Adresse des leeren Baumes ist 0,

die Adresse des Wurzelknotens ist 1,

ist die Adresse des aktuellen Knotens $z_1 \ldots z_k$, dann ist die Adresse des linkesten Kindknotens $z_1 \ldots z_k 0$, die Adresse des zweitlinkesten Kindknotens $z_1 \ldots z_k 1$ usw. (Bei mehr als 9 Kindknoten braucht man entsprechend viele Ziffern.)

Bei Binärbäumen bestehen die Ziffern nur aus 0 und 1. Daher sind Die Adressen einfache Bitfolgen. In eine Integervariablen mit 32 Bits passen dann Adressen von balancierten Bäumen mit bis zu 2^{31} Knoten, das sind Milliarden von Knoten.

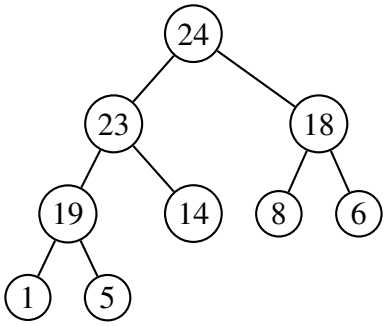


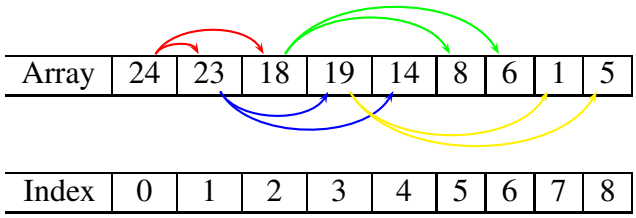
Ein binärer Baum mit Knotenadressen

2.5 Speicherung von Bäumen in Arrays

Ein binärer Baum kann linear in einem Array gespeichert werden, indem man die Schichten nacheinander abspeichert.

Für den folgenden Baum ist die Speicherung im Array rechts dargestellt.





Wenn das Array ab Index 0 gezählt wird, sind die Indices für die Kindknoten eines Knotens bei Arrayposition i: $2i + 1$ und $2i + 2$. Für das obige Beispiel bedeutet das:

Arrayindex	Arrayinhalt	Index der Kindknoten	Kindknoten
0	24	1,2	23,18
1	23	3,4	19,14
2	18	5,6	8,6
3	19	7,8	1,5

Von dem Arrayindex i eines Kindknotens kann man den Index des Elternknotens berechnen:

$$\mathit{Eltrnknotenindex}(i) = \lfloor (i-1)/2 \rfloor$$

Im Beispiel ergibt sich dabei:

Index des Kindknotens	0	1	2	3	4	5	6	7	8
Index des Elternknotens	0	0	0	1	1	2	2	3	3

Falls der Baum nicht vollständig ist, müssen die entsprechenden Arrayzellen gekennzeichnet werden.

Bäume vom Grad g kann man in gleicher Weise abspeichern.

Wenn wiederum das Array ab Index 0 gezählt wird, sind die Indices für die Kindknoten eines Knotens bei Arrayposition i dann: $g \cdot i + 1, \ldots, g \cdot i + g$.

Der Index des Elternknotens ergibt sich zu

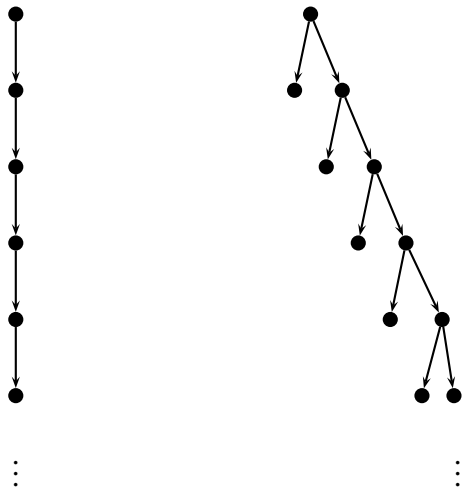
$$\mathit{Elternknotenindex}(i) = \lfloor (i-1)/g \rfloor$$

Auch wenn die Knoten des Baumes nicht Zahlen sind, dann kann man den Baum im Array abspeichern. Die Arrayeinträge wären denn Verweise (Pointer) auf die Knoten-Datenstrukturen.

2.6 Degenerierte Bäume

Alle Bäume in diesem Miniskript, außer denen in diesem Abschnitt, sind weitgehend *ausbalanciert*, d.h. alle Pfade bis zu den Blattknoten sind ungefähr gleich lang. In vielen Anwendungen ist es ganz wichtig, dass die Bäume ausbalanciert sind. Nur dann ist die Tiefe des Baumes gering genug, dass die Algorithmen daraus Vorteile ziehen können.

Dass das nicht immer so sein muss, zeigen die folgenden Beispiele:

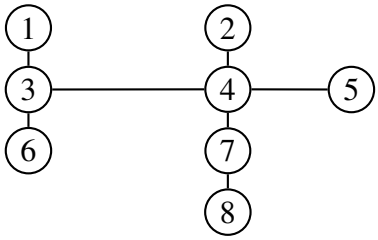


Degenerierte Bäume

In etlichen Anwendungen, z.B. AVL-Bäume oder B-Bäume macht man einigen Aufwand, um so degenerierten Bäume zu vermeiden.

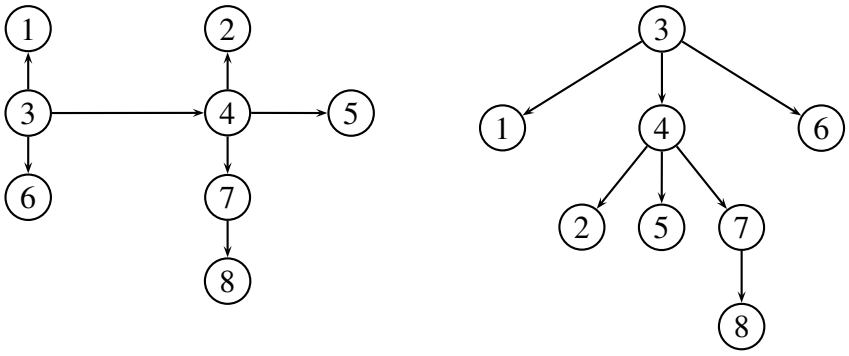
3 Ungerichtete Bäume

Zyklenfreie ungerichtete Graphen kann man ebenfalls als Bäume ansehen. Hier ist ein Beispiel:

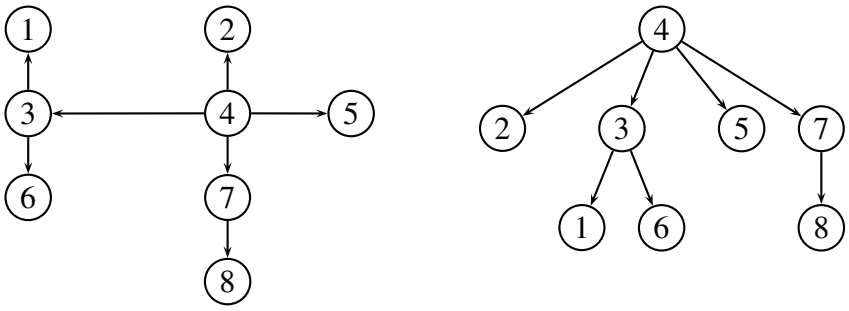


Die Blattknoten wären die Knoten, die nur mit einer Kante verbunden sind. Im Beispiel sind das die Knoten 1,2,5,6 und 8. Von den Blattknoten aus kann man die Darstellung des Baumes so umformen, dass sie tatsächlich wie ein Baum aussieht.

Im linken Baum unten haben wir den Kanten künstlich eine Richtung gegeben, und für diese Richtungen dann rechts den Baum in der üblichen Weise gezeichnet.



Das ist aber nicht die einzige Möglichkeit. Eine Alternative ist:



Stichwortverzeichnis