Bäume in der Informatik

Ein Leitprogramm von Matthias Niklaus

Inhalt und Lernziele:
Bäume gehören in der Informatik zu den wichtigsten Datenstrukturen. Dieses Leitprogramm verschafft einen Einblick in die Datenstruktur Baum.

Unterrichtsmethode: Leitprogramm
Das Leitprogramm ist ein Selbststudienmaterial. Es enthält alle notwendigen Unterrichtsinhalte, Übungen, Arbeitsanleitungen und Tests, die die Schüler/innen brauchen, um ohne Lehrperson lernen zu können.

Fachliches Review:
Juraj Hromkovic, Informationstechnologie und Ausbildung, ETH Zürich

Fachdidaktisches Review:
Juraj Hromkovic, Informationstechnologie und Ausbildung, ETH Zürich

Publiziert auf EducETH:
29. Mai 2007

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Verhaltenswissenschaft

Leitprogramm

Bäume in der Informatik

Fachgebiet Informatik

Gymnasium, Oberstufe
SchülerInnen im letzen Schuljahr

Bearbeitunsdauer: ca. 12 Lektionen

Autor: Matthias Niklaus
Betreuer: Prof. Juraj Hromkovic

Fassung vom: 29. Mai 2007
Noch keine Schulerprobung

Informatik
Computer Science

0.1 Einführung

Bäume gehören in der Informatik zu den wichtigsten Datenstrukturen. Der Baum wird sehr häufig für alle Arten von Problemen verwendet, wie zum Beispiel für die Suche, das Verwalten von Datenmengen, oder für die Speicherung einer geometrischen Struktur. Die Datenstruktur an sich ist sehr simpel, gibt dafür einen Raum für sehr viele Variationen und verschiedene Implementierungen.

Dieses Leitprogramm soll einen Einblick in die Datenstruktur Baum verschaffen und versucht die Vorteile von Bäumen anhand zweier spezieller Arten von Bäumen zu erläutern. Es werden nicht nur die theoretischen Grundlagen vermittelt, sondern es soll gleich noch eine übersichtliche Implementation der Datenstruktur aufgezeigt werden.

Bäume als Datenstruktur können noch beliebig weiter vertieft werden über den Rahmen dieses Leitprogrammes hinaus. Effizienz von balancierten Bäumen, Laufzeitanalyse und andere Arten von Bäume für andere Probleme könnten noch weiteres Lernmaterial bieten.

0.1.1 Vorkenntnisse

- Die Klasse weiss was allgemeine Graphen sind. Bäume werden als ein spezieller Graph gesehen.
- Die Schülerinnen und Schüler sollten bereits gewisse grundlegende Datensrukturen der Informatik kennen, wie zum Beispiel verkettete Listen, Stack (Stapel) und Queue (Schlange).
- Das Prinzip der Rekursion sollte bekannt sein.
- Da in den praktischen Übungen Java-Code vorgegeben und Java programmiert wird, sollten Sie bereits Vorkenntnisse in Java haben. Falls eine ähnliche objekt-orientierte Programmiersprache bekannt ist, müssen die Unterschiede erläutert werden, oder der vorgegebene Programmcode abgeändert werden.
  Folgende höhere Elemente aus Java oder OO werden benötigt.
  - Vererbung und Polymorphismus
  - Abgeschlossenheit (Encapsulation)
  - Abstrakte Klassen
  - Das *static* Schlüsselwort
- Die Klasse sollte die Infix- Präfix- und Postfix-Darstellungen von mathematischen Ausdrücken kennen und wissen, wie sie interpretiert (ausgewertet) werden.

Inhaltsverzeichnis

0.2 Arbeitsanleitung

Für die Bearbeitung dieses Leitprogramms sind zwei unterschiedliche Umgebungen vorgesehen. Ein Teil der Kapitel besteht aus der theoretischen Erarbeitung des Stoffes, der andere Teil enthält die praktische Anwendung des Gelernten. Darum benötigen Sie einerseits einen Arbeitsraum mit Tischen, aber auch einen Computerraum, wo die praktischen Übungen stattfinden können.

Die theoretischen Aufgaben sollen als Einzelarbeit gelöst werden. Für die praktischen Aufgaben am Computer ist Gruppenarbeit vorgesehen.

Die Kapitel sollten prinzipiell in der vorgegebenen Reihenfolge abgearbeitet werden, da in den späteren Kapitel auf das Wissen aus den früheren Kapiteln Bezug genommen wird. Falls aber Mangel an Computerplätzen herrscht, kann das theoretische Kapitel 4 vor dem praktischen Kapitel 3 bearbeitet werden, und so die Klasse aufgeteilt werden.

Kapitel 1

Einführung

1.1 Übersicht

1.1.1 Was lernen Sie hier?

In diesem Kapitel sollen Sie im ersten Teil ein Verständnis von Bäumen in der Informatik erhalten. Mit mehreren Beispielen versuchen wir eine gemeinsame Vorstellung über Bäume in der Informatik zu erlangen. Anschliessend werden erste Anwendungen von Bäumen in der Informatik gezeigt.

Im zweiten Teil lernen Sie notwendige Begriffe. Diese Begriffe benötigen wir, um über Bäume in der Informatik sprechen zu können. Es ist wichtig, immer klar zu wissen, was ein Begriff genau bedeutet.

1.2 Lernziele

Nachdem Sie dieses Kapitel durchgelesen haben,

haben Sie eine Vorstellung was man unter Bäumen in der Informatik versteht.

kennen Sie erste Anwendungen für Bäume in der Informatik.

kennen Sie einige Begriffe für den Umgang mit Bäumen. Diese können Sie auch einem interessierten Kollegen erklären.

wissen Sie wie Bäume definiert sind.

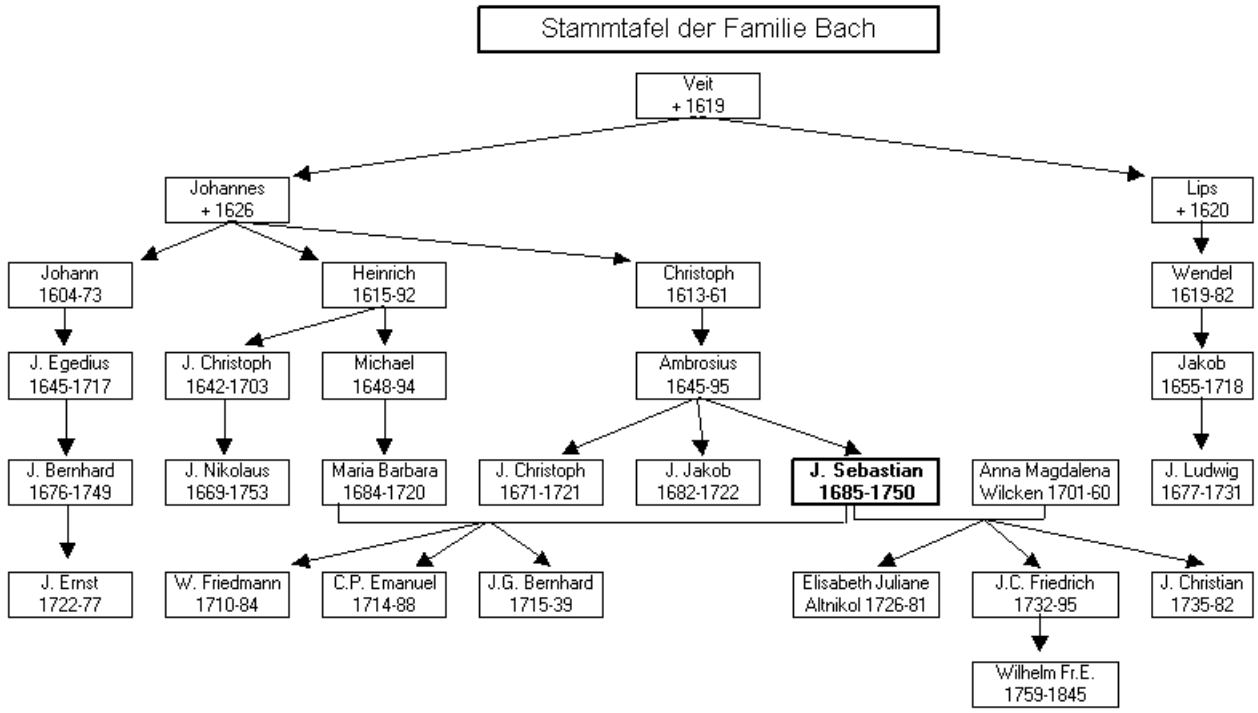


Abbildung 1.1: Ein Stammbaum, Beispiel für Bäume als Datenstrukturen.

1.3 Konzept von Bäumen

Bäume gehören nicht nur in der Informatik zu den wichtigsten Datenstrukturen. Eine bekannte baumartige Datenstruktur ist beispielsweise der Stammbaum.

Im Stammbaum wird auf jeder Stufe eine Generation dargestellt. Wählt man eine Person aus einer Generation aus, so befinden sich deren Eltern direkt eine Stufe darüber und die Kinder eine darunter. Oft sind die Kinder und die Eltern mit einer Linie verbunden. Es ist also eine hierarchische Struktur. Es gibt zwei verbreitete Arten von Stammbäumen. Der Ahnenbaum weist alle Vorfahren eines Menschen oder Paares auf. Die zweite Art listet alle Nachfahren eines Menschen oder Paares auf. Im weiteren wollen wir nur noch von dieser zweiten Art von Stammbäumen sprechen. Hier steht zuoberst die Ausgangsperson, von der alle Nachfahren aufgezeigt werden. Diese Ausgangsperson bildet die Wurzel des Baumes. Auf der Abbildung 1.1 ist gut zu sehen, dass dieser Baum nach unten wächst. Natürlich könnte man den Baum horizontal spiegeln, so dass der Baum nach oben wächst. Diese Darstellung würde dann auch mit den Bäumen aus der Natur übereinstimmen. Es hat sich aber gezeigt, dass es viel intuitiver ist, die abstrakten Datenbäume nach unten wachsen zu lassen. Dies kann damit zusammenhängen, dass wir im allgemeinen immer oben mit dem Betrachten von Dokumenten beginnen. So passt es, wenn die Ausgangsperson ganz oben positioniert ist.

Ein weiteres Beispiel für eine baumartige Datenstruktur ist ein Führungsdiagramm ei-

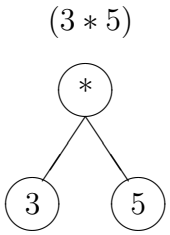


Abbildung 1.2: Ein mathematischer Ausdruck repräsentiert als Baum.

nes Unternehmens. Auch dieses Beispiel zeigt eine Hierarchie auf. Der Verwaltungsrat steht zuoberst und trägt die gesamte Verantwortung. Direkt unter dem Verwaltungsrat ist der CEO. Dem CEO sind in unserem Beispiel der CFO (Finanzchef), CIO (Informatikchef) und der CPO (Einkaufsleiter) unterstellt. Weiter hat der CPO wieder 5 Abteilungsleiter die ihm direkt unterstellt sind. Auch für den CFO und den CIO gibt es wieder direkt Unterstellte. So entsteht wieder die gleiche Struktur wie beim Stammbaum. Das Führungsdiagramm geht immer weiter bis es bei der untersten Stufe angelangt ist. Zwischen dem CFO und den Unterstellten des CPO gibt es keine direkte, eindeutige Beziehung. Hat der CFO etwas mit einem Untergebenen des CPO zu klären, muss dies über den CPO gehen. Dies ist so, weil der CFO nicht in der Hierarchieleiter des betroffenen Angestellten ist.

Aufgabe 1.1 *Zeichnen Sie den Baum für das oben beschriebene Führungsdiagramm. Verwenden Sie den Stammbaum in Abb. 1.1 als Vorlage.*

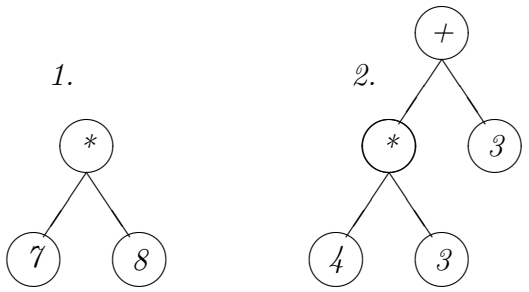
Aufgabe 1.2 *Sie haben hier zwei bekannte Beispiele für Bäume gesehen. Suchen Sie noch weitere Beispiele bei denen Daten oder Strukturen in der Form eines Baumes abgebildet werden.*

Was für Anwendungen haben wir in der Informatik für solche Bäume? Diese Frage sollte in diesem Abschnitt beantwortet werden.

Die am meisten verbreitete Anwendung von Bäumen sind die so genannten Suchbäume. Suchbäumme können verwendet werden, um rasch zu überprüfen, ob eine bestimmte Zahl in einer Menge von Zahlen vorhanden ist. Eine weitere häufige Anwendung von Bäumen ist das darstellen von mathematischen Ausdrücken oder ganzen Computerprogrammen. Die Abbildung 1.2 zeigt einen mathematischen Ausdruck als Baum dargestellt. Das interessante bei dieser Darstellung ist, dass die Klammerung des Ausdruckes nicht explizit nötig ist. Die Auswertungsfolge wird vom Baum vorgegeben. Jeder Eintrag, der weitere Nachkommen hat, ist ein Operator. Die Nachfolger sind die Operanden. Es gibt immer so viele Nachfolger wie ein Operator Operanden hat. Ein Operator kann ausgewertet werden, indem die Operation auf den Operanden angewendet wird. Falls ein Nachfolger wieder ein Operator ist, so muss zuerst dieser Operator ausgewertet werden, bevor sein Vorfahre ausgewertet werden kann.

Aufgabe 1.3 *Zeichnen Sie die Bäume für die mathematischen Ausdrücke* $(3 * 2)$ *und* $(4 + 3)$.

Aufgabe 1.4 *Wandel Sie die folgenden Bäume in mathematische Ausdrücke um.*



1.4 Begriffe und Definitionen

Jetzt haben Sie eine erste Vorstellung der Baumstruktur. Nun benötigen wir Begriffe, damit wir präzise über die Bäume sprechen können. Im folgeneden Abschnitt werden die grundlegenden Begriffe für Bäume eingeführt.

Begriffe

Bäume in der Informatik bestehen aus Elementen, die eine baumartige Beziehung untereinander haben. Diese Elemente nennen wir **Knoten**. Bei einem Stammbaum entspricht jede Person einem Knoten. Alle Knoten bis auf einen haben einen Vorfahren. Dieser eine Knoten ohne Vorfahren nennen wir **Wurzel**. Die Wurzel wird, wie beim Stammbaum, meist als oberster Knoten dargestellt. Alle direkten Nachfolger der Wurzel sind die **Kinder** der Wurzel. Der Vorfahre eines Knoten wird **Vater** genannt. Vielleicht wird man den Vater in Zukunft auch Mutter oder Elter nennen. Diese Begriffe sind dual zu denjenigen in einem Stammbaum. Jeder aufgezeichneter Stammbaum ist endlich. Das heisst, es gibt immer Personen die keine Kinder haben. Das gleiche gilt auch für die Bäume in der Informatik. So gibt es Knoten die keine Kinder haben. Diese Knoten nennen wir **Blätter**. Der Begriff Blatt lässt sich sehr gut von den natürlichen Bäumen übertragen. Die Blätter bilden immer den Abschluss eines Baumes. Innen sind die Äste, die sich immer weiter verzweigen bis die Blätter den Abschluss bilden. Jetzt haben wir drei verschiedene Arten von Knoten. Die Wurzel, die Blätter und alle anderen Knoten, die wir **innere Knoten** nennen.

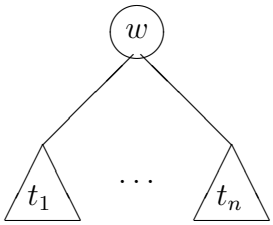


Abbildung 1.3: Baum mit Wurzel w mit den Teilbäumen $t_1 \ldots t_n$.

Aufgabe 1.5 *In der Zusammenstellung sind alle eingeführten Begriffe aufgelistet. Schreiben Sie zu jedem Begriff eine Definition in Ihren eigenen Worten. Versuchen Sie sich möglichst genau auszudrücken.*

- *Knoten*
- *Wurzel*
- *innerer Knoten*
- *Blatt*
- *Vater*
- *Kind*

Definitionen

Das waren schon alle Bestandteile für unsere Bäume. Wurzel, Blätter und innere Knoten bilden die Elemente und die Vater- und Kind-Beziehungen bilden die Relationen zwischen den Knoten.

Wir wollen hier die Datenstruktur Baum rekursiv definieren. Rekursiv bedeutet, dass die Definition immer wieder mit sich selbst erweitert werden kann. Da unsere Definition auf der Erweiterung beruht, ist es sinnvoll, mit dem kleinsten Baum zu beginnen. So können wir alle anderen Bäume, die grösser sind, daraus aufbauen.

Rekursive Definition

Ein Baum T ist entweder leer oder besteht aus einer Wurzel w mit $0 \ldots n$ Teilbäumen $t_1 \ldots t_n$.

Erklärung der rekursiven Definition. Der kleinste nicht leere Baum besteht aus nur einem Knoten w. Dieser Knoten ist Wurzel und Blatt in einem. Haben wir $t_1, ..., t_n$ weitere Bäume, so können wir die Wurzeln von $t_1, ..., t_n$ zu Kindern von w machen. Somit erhalten

wir einen neuen Baum, der aus der Wurzel w und den Bäumen $t_1, ..., t_n$ besteht. Den Baum t_i $(1 \leq i \leq n)$ $i - ter$ nennen wir Teilbaum der Wurzel w, da er nun ein Teil des neuen Baumes ist. Die Abb. 1.3 zeigt den zusammengesetzten Baum mit der Wurzel w. Mit diesem Schema lassen sich alle vorstellbaren Bäume erstellen.

Aufgabe 1.6 *Sie kennen die Liste als Datenstruktur. Die Liste besteht aus Elementen. Ein Element ist als Start-Element ausgezeichnet. Danach können beliebig viele Elemente folgen. Wenn kein Element mehr folgt, dann ist die Liste am Ende. Versuchen Sie für die Liste eine rekursive Definition anzugeben. Verwenden Sie die Definition des Baumes als Vorlage.*

Nicht rekursive Definition

Ein Baum T=(V,E) besteht aus einer Menge von Knoten V und einer Menge von gerichteten Kanten E. Die Wurzel $w \in$ V hat nur Ausgangskanten. Alle anderen Knoten haben genau eine Eingangskante. Für alle Kanten $e \in$ E gilt $e =$ (v1,v2), wobei v1,v2 $\in$ V und v1 $\neq$ v2.

Erklärung der nicht rekursiven Definition. Diese Definition kommt aus der Graphentheorie. Der Baum wird hier als ein besonderer Graph beschrieben. Der nicht leere Baum besteht aus der Wurzel w. Die Wurzel ist dadurch gekennzeichnet, dass sie nur ausgehende Kanten hat. Alle anderen Knoten im Baum haben genau eine eingehende Kante. Der Baum ist ein zusammenhängender Graph. Die Kanten verbinden immer zwei verschiedene Knoten. Diese Bedingungen reichen aus, um die gleiche Struktur wie von der rekursiven Definition zu erzwingen.
Die Richtung der Kanten kann mit einem Pfeil bestimmt sein. Bei den meisten Bäumen wird kein Pfeil verwendet. Bei den Bäumen ohne Pfeile verlaufen die Kanten von oben nach unten. Die Abb. 1.2 zeigt einen Baum, mit Kanten die von oben nach unten verlaufen.

Aufgabe 1.7 *Betrachten Sie die nicht rekursive Definition genau. Sie sollen nun im gleichen Stil eine Definition für die Datenstruktur Liste erarbeiten.*

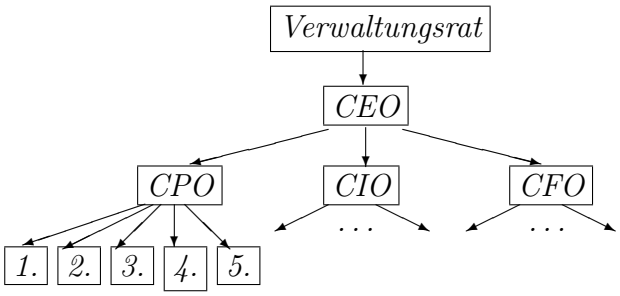
Bei der rekursiven Definition hat die Wurzel w n Kinder, wobei n eine beliebige natürliche Zahl sein kann. Die Bäume $t_1, ..., t_n$ können nur kleinste Bäume aus einem Knoten oder Bäume, die mit obiger Definition erzeugt wurden, sein. Daraus folgt, dass es im ganzen neu erzeugten Baum keinen Knoten gibt, der mehr wie n Kinder haben kann. So sagen wir der Baum habe die Ordnung n. Die **Ordnung** eines Baumes ist die maximale Anzahl Kinder die ein Knoten in diesem Baum haben kann.

Es gibt eine noch speziellere Definition von Bäumen. Diese verlangt, dass in einem Baum der Ordnung n jeder Knoten genau n oder 0 Kinder hat. Mit dieser Definition ist es nun nicht mehr möglich beliebige Bäume zu erzeugen. Damit die Vollständigkeit erhalten bleibt, führt man hier aber meist noch den leeren Baum ein, der aus 0 Knoten besteht.

Der leere Baum ist per Definition ein Baum jeglicher Ordnung. Nun kann wieder jeder beliebiger Baum erzeugt werden, indem man für fehlende Teilbäume einen leeren Baum einfügt.

1.5 Lösungen zu den Aufgaben

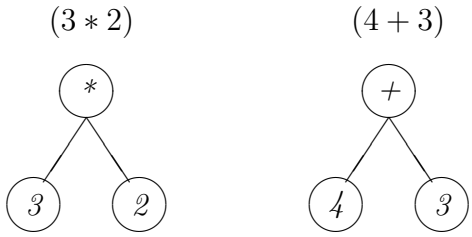
Antwort 1.1 *Der folgende Baum stellt das im Beispiel beschriebene Führungsdiagramm dar. Ihre Abbildung kann von diesem Baum abweichen.*



Antwort 1.2 *Weitere Beispiele für baumartige Strukturen sind:*

- *Die Kirchenstruktur*
- *Das UNO Organisationsdiagramm*
- *Ein mathematischer Ausdruck als Baum*
- *Eine Spitalordnung (Chefarzt, Oberarzt, Unterarzt, Assistenzarzt, etc.)*

Antwort 1.3 *Die Bäume stellen die beiden Ausdrücke dar. Da diese mathematischen Ausdrücke kommutativ sind, gibt es noch weitere Arten diese darzustellen.*



Antwort 1.4 *Die beiden Bäume lassen sich auch auf verschiedene Arten als Ausdrücke schreiben.*

1. $(7 * 8)$
2. $((4 * 3) + 3)$

Antwort 1.5 *Diese Lösung kann nicht mit Ihrer überein stimmen, doch sollte sie die Möglichkeit geben, ihre Definitionen zu überprüfen.*

- *Knoten: Ein Knoten ist ein Element in einem Baum.*
- *Wurzel: Eine Wurzel ist der oberste Knoten eines Baumes. Sie hat keinen Vater.*
- *Innererer Knoten: Ein innerer Knoten ist ein Knoten der Kinder und Vater hat. Ein Knoten der nicht Blatt und nicht Wurzel ist.*
- *Blatt: Ein Blatt ist ein unterster Knoten eines Baumes. Es hat keine Kinder mehr.*
- *Vater: Ein Vater ist der Vorfahre eines bestimmten Knotens.*
- *Kind: Ein Kind ist der Nachkomme eines bestimmten Knotens.*

Antwort 1.6 *Die rekursive Definition einer Liste ist relativ simpel. Die Idee ist folgende: Es gibt eine leere Liste. Dann sagen wir: Eine Liste ist entweder eine leere Liste oder ein Konstrukt, das aus einem Element und einer an das Element angehängten Liste besteht. Wie wir sehen, ist die Definition mit sich selbst erweitert, da das Wort* Liste *in der Definition von Liste wieder vorkommt, das heisst, sie ist rekursiv.*

Antwort 1.7 *Die nicht rekursive Definition einer Liste. Die Idee ist folgende: Die Liste ist ein zusammenhängender Graph L=(V,E) der aus einer Menge von Knoten V und gerichteten Kanten E besteht. Alle Knoten haben maximal eine ausgehenden Kante. Das erste Element in der Liste hat keine eingehende Kante. Alle anderen Knoten haben genau eine eingehende Kante. Für alle Kanten* $e \in E$ *gilt* $e = (v1,v2)$*, wobei* $v1,v2 \in V$ *und* $v1 \neq v2$.

Kapitel 2

Konzept Etiketten- und Ausdrucks-Bäume

2.1 Übersicht

2.1.1 Was lernen Sie hier?

Sie kennen jetzt die Grundstruktur und schon einige Anwendungsbeispiele von Bäumen. In diesem Kapitel wollen wir die Anwendungen von Etiketten- und Ausdrucks-Bäumen betrachten.

Durch die Betrachtung einiger Beispiele werden wir Anforderungen für diese Bäume erarbeiten. Diese Anforderungen müssen von einer Implementation erfüllt werden.

Das intuitive Verständnis und die Liste der Anforderungen für die Etiketten- und Ausdrücks-Bäume liefert die Grundlage für das nächste Kapitel.

2.2 Lernziele

Nachdem Sie dieses Kapitel durchgelesen haben,

kennen Sie verschiedene Anwendungen für Etiketten- und Ausdrucks-Bäume.

wissen Sie, wie man Bäume traversieren kann.

können Sie die Anforderungen für eine Implementation eines Etiketten- oder Ausdruck-Baumes erarbeiten.

2.3 Etiketten-Bäume

Etiketten-Bäume sind Bäume, bei denen jeder Knoten eine Etikette hat. Diese Etiketten können mit irgendwas beschriftet sein. Etiketten-Bäume werden häufig auch *labeled trees* genannt. Das Wort *label* bedeutet Beschriftung oder eben Etikette.
Die Beschriftung der Etikette kann sehr allgemein aufgefasst werden. So muss die Beschriftung nicht Text sein, sondern kann ein beliebiges Objekt sein.

Etiketten-Bäume werden verwendet, um baumartige Strukturen darzustellen. Die meisten Beispiele aus dem Kapitel 1 sind offensichtliche baumartige Strukturen. Es ist naheliegend, die natürliche Struktur der Daten beizubehalten. Hier die Liste mit den baumartigen hierarchischen Strukturen aus Kapitel 1.

- Inhaltsverzeichnis
- Mathematische Ausdrücke
- Programmcode
- Stammbaum
- Firmen Führungsdiagramme

Im nächsten Abschnitt soll die Struktur dieser Daten anhand eines Beispiels aufgezeigt werden.

2.3.1 Die Struktur als Information

Betrachten wir kurz ein Inhaltsverzeichnis eines Algorithmen-Buches. Hier ein Auszug von üblichen Einträgen.

Motivation
Schlussfolgerung
Baumalgorithmen
Graphenalgorithmen
Tiefensuche
Tiefensuche
Verbesserte Tiefensuche

Aus dieser Auflistung geht nicht viel hervor. Das Buch behandelt Graphen- und Baumalgorithmen. Zu einem Thema gibt es eine Motivation und es hat auch eine Schlussfolgerung. Die Tiefensuche wird zwei mal besprochen und es gibt auch noch eine verbesserte Tiefensuche.
Der Eintrag “Motivation“ kann vieles Bedeuten. Er könnte eine Motivation für das ganze Buch oder auch die Motivation für ein bestimmtes Kapitel ankündigen. Der Eintrag “Verbesserte Tiefensuche“ bezieht sich offensichtlich auf einen im Buch vorgestellten Algorithmus für die Tiefensuche. Es ist aber von Interesse, zu welchem Algorithmus es eine

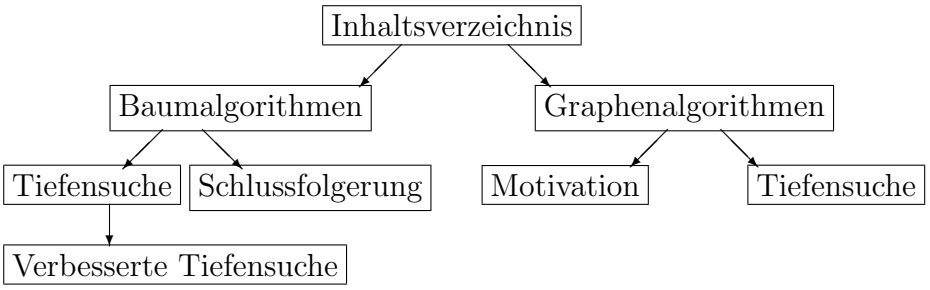


Abbildung 2.1: Inhaltsverzeichnis als Baum dargestellt.

Verbesserung gibt.
Ich gebe zu, dass ich absichtlich etwas Unordnung in die Auflistung brachte. Es ist aber nun klar, dass in einem Inhaltsverzeichnis nicht nur die Einträge zählen, sondern auch die Anordnung der Einträge. In Abb: 2.1 ist das Inhaltsverzeichnis als Baum mit Boxen und Pfeilen dargestellt. Es ist sofort zu erkennen, dass es für die Baumalgorithmen eine verbesserte Tiefensuche gibt. Die Motivation ist für das Graphen-Kapitel. Dies war zu erwarten, da die Baumalgorithmen natürlich viel spannender sind. :-)
Die Unterkapitel sind die Kinder der übergeordneten Kapitel. Die Anordnung der Kinder ist auch wichtig. Die Motivation ist das erste Unterkapitel bei den Graphenalgorithmen. Die Tiefensuche ist dann das zweite Unterkapitel direkt nach der Motivation.

Betrachten wir kurz das Inhaltsverzeichnis in einer üblicheren Form.

- Inhaltsverzeichnis
  1. Baumalgorithmen
     (a) Tiefensuche
        i. Verbesserte Tiefensuche
     (b) Schlussfolgerung
  2. Graphenalgorithmen
     (a) Motivation
     (b) Tiefensuche

Die Struktur des Buches wird sofort klar. Wenn man das Inhaltsverzeichnis mit Abb: 2.1 vergleicht, sieht man die übereinstimmende Struktur. Somit ist dies auch eine Darstellung für einen Baum.
Wichtig bei allen obigen Anwendungen ist, dass die Struktur richtig abgebildet wird. Die Struktur ist ein entscheidender Bestandteil der Daten.
Übertragen wir dies auf die Bäume. Die Informationen, die wir speichern, sind nicht nur die Etiketten an den Knoten, sondern auch die Position eines Knoten.

Aufgabe 2.1 *Nehmen Sie eines Ihrer Lehrbücher und stellen Sie das Inhaltsverzeichnis als einen Baum mit Boxen und Pfeilen wie in Abb:2.1 dar. Verwenden Sie ein Buch mit einem vernünftig grossen Inhaltsverzeichnis. Sie sollten nicht mehr wie 30 Boxen malen müssen.*

2.3.2 Die Etiketten versus Namen

Die Knoten haben immer einen eindeutigen Namen. Die Etiketten verschiedener Knoten dürfen den gleichen Inhalt haben. Knotennamen und Etiketten sind zwei unabhängige Eigenschaften eines Knotens. In Abb: 2.1 ist ein Inhaltsverzeichnis als Etiketten-Baum dargestellt. Die Knoten sind in dieser Abbildung nur mit dem Inhalt der Etiketten beschriftet. Dies reicht aus, solange wir nicht über bestimmte Knoten sprechen wollen. Soll ein Baum diskutiert werden, müssen die Knoten auch mit ihren Namen gekennzeichnet sein. So ist immer klar, von welchem Knoten gesprochen wird.

Was kann alles auf einer Etikette stehen?
Grundsätzlich kann alles einer Etikette zugeordnet werden. Bei den Beispielen aus Kapitel 1 sind es immer Zeichenketten. In der Informatik werden meistens Objekte einer bestimmten Klasse dem Knoten zugeordnet. Aber es kann wirklich jedes Objekt einem Knoten zugewiesen werden. So ist es vorstellbar, einen Etiketten-Baum zu bauen, wo jedes Etikett wieder einem Etiketten-Baum entspricht. Die Etiketten sind die Daten, die wir an der Position des Knoten Speichern wollen.

An zwei Beispielen wollen wir die Anforderungen für Etiketten-Bäume erarbeiten. Zuerst betrachten wir ein Organisationsdiagramm eines Unternehmens, danach betrachten wir mathematische Ausdrücke.

2.4 Organisationsdiagramme

Es ist wichtig, aus welchen Leuten die Führungsmannschaft einer Organisation zusammengestellt ist. Damit die Führung funktionieren kann, muss sie organisiert sein. Die Führungsmannschaft wird strukturiert. Wer ist der Vorgesetzte von Frau Meier? Das ist zum Beispiel eine Frage, die hier wichtig scheint. Der Personalchef muss auch alle Untergebenen von Frau Meier finden können, um diese zu informieren falls er Frau Meier ersetzt. Ein Organisationsdiagramm ist meistens als Baum strukturiert. Jedes Kadermitglied hat eine Gruppe direkter Untergebener. Alle Kinder vom Knoten *Frau Meier* entsprechen den Untergebenen von Frau Meier. Hier ist zu beachten, dass nicht die leiblichen Kinder gemeint sind, sondern die Kinder im Baum. Abb. 2.2 zeigt einen Organisationsbaum. Der Knoten mit dem Namen k1 entspricht der obersten Position in der Organisation. Auf dem Etikett des Knoten k1 steht Frau Meier. Daraus folgt, dass Frau Meier die höchste Position inne hat. Wie oben beschrieben sehen wir, dass Frau Meier vier direkte Untergebene hat. Wird eine Untergebene ersetzt, dann muss nur die Etikette ausgewechselt werden. Die Baumstruktur wird nicht abgeändert. So ist der Aufstieg einer Person nicht mit Änderungen der Struktur verbunden.
Wird eine neue Abteilung gegründet, dann bekommt Frau Meier einen neuen Untergebenen. Jetzt muss der Baum angepasst werden. Der Knoten k1 bekommt ein weiteres Kind. Dieser neuer Knoten entspricht nun der neu geschaffenen Position in der Organisation. Das Etikett kann mit einer beliebigen Person beschriftet werden. Die Etikette könnte in diesem Beispiel ein Objekt vom Typ `Mitarbeiter` sein. Der Typ `Mitarbeiter` enthält alle

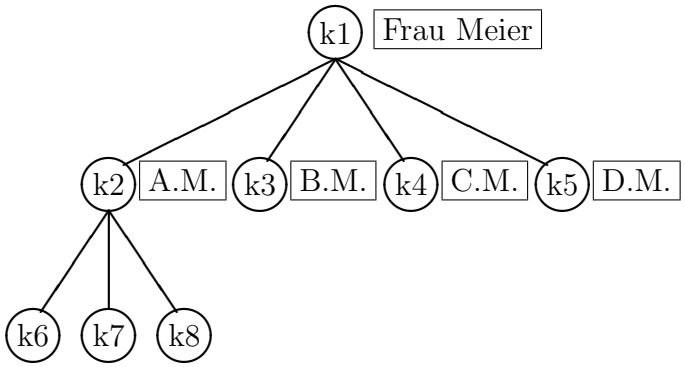


Abbildung 2.2: Ein Organisationsbaum mit den zugehörigen Etiketten.

relevanten Eigenschaften der Person.

2.4.1 Anforderung

Hier diskutieren wir die Anwendungen die ein Organisationsdiagramm unterstützen sollte. Daraus wollen wir die Anforderungen für den Baum erarbeiten, der ein Organisationsdiagramm repräsentiert.

Zur Abbildung der Struktur muss der Baum für jede Position im Organisationsdiagramm einen Knoten haben. Die Position der Knoten darf sich nicht verändern, ausser die Struktur wird abgeändert. Da die Anzahl der Untergebenen nicht fixiert ist, muss der Baum für jeden Knoten eine beliebige Anzahl von Kindern unterstützen. Die Vater- Kind-Beziehung der Knoten muss immer erhalten bleiben, damit sich die Position der einzelnen Knoten nicht verändert.

Welche Operationen muss man ausführen können? Von einer gegebenen Position sollten einfach die Vorgesetzten und die Untergebenen gefunden werden. Daraus folgt für den Baum: Ist ein Knoten gegeben, so sollte der Vater und die Kinder einfach bestimmt werden können. Die Führungsorganisation ändert nicht sehr oft, aber sie kann ändern. Deshalb muss es möglich sein, einzelene Knoten zu entfernen oder neu hinzuzufügen. Diese Operation muss nicht besonders effizient sein, da sie selten benötigt wird.

Aufgabe 2.2 *Überlegen Sie sich, ob es noch weitere Anforderungen an ein Organisationdiagramm gibt. Reichen die Anforderungen, die wir an den Baum gestellt haben, aus, um auch Ihren neue Anforderungen gewachsen zu sein?*
Ist dies nicht der Fall, dann teilen Sie die zusätzlichen Anforderungen ihrem Betreuer mit, damit sie in das Leitprogramm aufgenommen werden können.

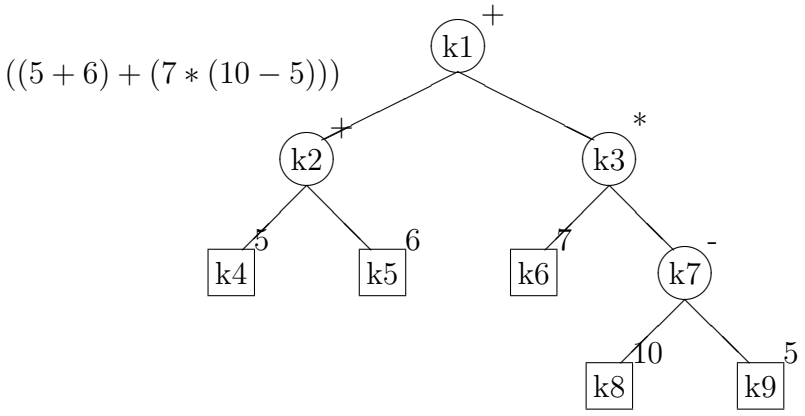


Abbildung 2.3: Mathematischer Ausdruck und zugehöriger Baum.

2.5 Mathematische Ausdrücke und Programmcode

In Kapitel 1 haben Sie schon einfache mathematische Ausdrücke als Bäume gesehen. Ist Ihnen dabei aufgefallen, ob die Anzahl Kinder für jeden Knoten gleich ist? Oder von was die Anzahl der Kinder eines Knoten abhängt?
Abb. 2.3 zeigt einen Etiketten-Baum, der den mathematischen Ausdruck $((5 + 6) + (7 * (10 - 5)))$ darstellt. Die Operatoren und Operanden sind auf die Etiketten geschrieben. Die Blätter des Baums entsprechen den Operanden, das heisst den Konstanten und Variablen. Alle anderen Knoten stellen Operatoren dar. In den einfachen Ausdrücken haben alle Knoten zwei Kinder. Dies liegt daran, dass die einfachen Ausdrücke nur aus Operatoren bestehen die zwei Operanden benötigen. Beispiele hierfür sind $\times$, $\div$, $+$ und $-$. Der Aufbau ist Unterausdruck - Operator - Unterausdruck. Die Summenformel *sum* ist ein Beispiel für einen Operator, der mehr als zwei Operanden benötigt. Die Summe besteht aus den zwei Grenzen und dem Ausdruck, von dem die Summe berechnet werden soll.

Die Operatoren $-$ und $\div$ sind nicht kommutativ, deshalb benötigen die Unterausdrücke eine Ordnung. Das bedeutet, dass die Unterbäume eines Knotens auch geordnet sein müssen. Dies erreichen wir, indem wir verlangen, dass die Kinder eines Knoten geordnet sind. Wollen wir einen Ausdruck darstellen, der eine Summe ($\sum$) beinhaltet, dann stehen wir vor einem Problem. Der Summen Operator müsste drei Kinder haben. Wie bei den Organisationsdiagrammen können wir wieder sagen, dass jeder Knoten beliebig viele Kinder haben kann. Oder wir legen für jeden Operator einzeln die Anzahl der Kinder fest. Beide vorgeschlagenen Mechanismen würden funktionieren, sind aber nicht einfach zu verwirklichen. Bäume sind viel einfacher zu implementieren, wenn sie immer die gleiche Anzahl Kinder haben. Für die mathematischen Ausdrücke wollen wir uns auf zwei Kinder beschränken. Ein weiterer wichtiger Begriff im Zusammenhang mit Bäumen ist der Binärbaum. Dieser kommt in der Informatik am häufigsten vor. Ein **Binärbaum** ist ein Baum, bei dem jeder Knoten maximal zwei Kinder hat. Also genau die Eigenschaft, die wir hier wollen.

Infix: $((5 + 6) + (7 * (10 - 5)))$
Präfix: $+ + 5\ 6 * 7$ - $10\ 5$
Postfix: $5\ 6 + 7\ 10\ 5$ - $* +$

Abbildung 2.4: Ein mathematischer Ausdruck in verschiedenen Notationen.

Aufgabe 2.3 *Da wir uns auf zwei Kinder beschränken wollen, müssen wir uns einen Trick überlegen. Versuchen Sie eine Möglichkeit zu finden, wie Sie den Summen- Operator repräsentieren können, so dass er nur zwei Kinder benötigt. Verwenden Sie nicht zu viel Zeit für ihre Suche. Es gibt eine simple Lösung, auf die man jedoch nicht immer gleich kommt.*
Tipp: Führen Sie einen Hilfs-Operator ein.

Haben Sie eine Lösung für obige Frage gefunden? Ist Ihnen bei der vorgeschlagenen Antwort etwas aufgefallen? Wie soll man erkennen, welches die untere und welches die obere Grenze ist? In der Lösung wurde die Reihenfolge der Grenzen nicht festgelegt. Es ist wichtig für jeden nicht kommutativen Knoten festzulegen, welche Reihenfolge die Kinder haben. Bei den binären Bäumen hat es sich eingebürgert, von linkem und rechtem Kind zu sprechen. Links entspricht dem ersten Kind.

Aufgabe 2.4 *Die Anordnung der Knoten in einem Baum sollte oft festgelegt sein. Zeichnen Sie einen Stammbaum Ihrer Familie, beginnend bei einem Ihrer Grossväter wie in Abb. 1.1. Sie sollen nur den Namen der Person eintragen. Es sollte einem Betrachter leicht möglich sein, aus der Struktur des Baumes zu erkennen, ob ein Kind jünger oder älter wie ein Geschwister ist.*
Schreiben Sie zu Ihrem Stammbaum die Annahmen die Sie getroffen haben.

Damit wir weitere Anforderungen an die Datenstruktur stellen können, müssen wir möglichst genau wissen, was wir mit den mathematischen Ausdrücken alles machen wollen.

Die häufigste Anwendung mathematischer Ausdrücke ist deren Berechnung. Sind die Ausdrücke in einer Datenstruktur erfasst, müssen sie einem Auswerter übergeben werden. Die maschinelle Auswertung mathematischer Ausdrücke ist eine alte Technik. So gibt es auch schon eine grosse Anzahl von Geräten, die das einigermassen beherrschen. Leider haben nicht alle Geräte das gleiche Eingabeformat. Die heutige Taschenrechner verstehen, die für uns übliche Infix-Notation. Die Abb. 2.4 zeigt die drei verschiedenen Notationen. Die HP Taschenrechner benötigten hingegen lange die Postfix-Notation. Es ist auch durchaus möglich, dass die Präfix-Notation verlangt wird.

Aufgabe 2.5 *Sie haben die verschiedenen mathematischen Notationen für Ausdrücke sicher schon kennen gelernt. Im Alltag verwenden wir eigentlich nur die Infix-Notation. In dieser Übung sollen Sie die Präfix und Postfix-Notation wieder etwas einüben. Wandeln Sie die folgenden mathematischen Ausdrücke in jeweils die zwei anderen Notationen um.*

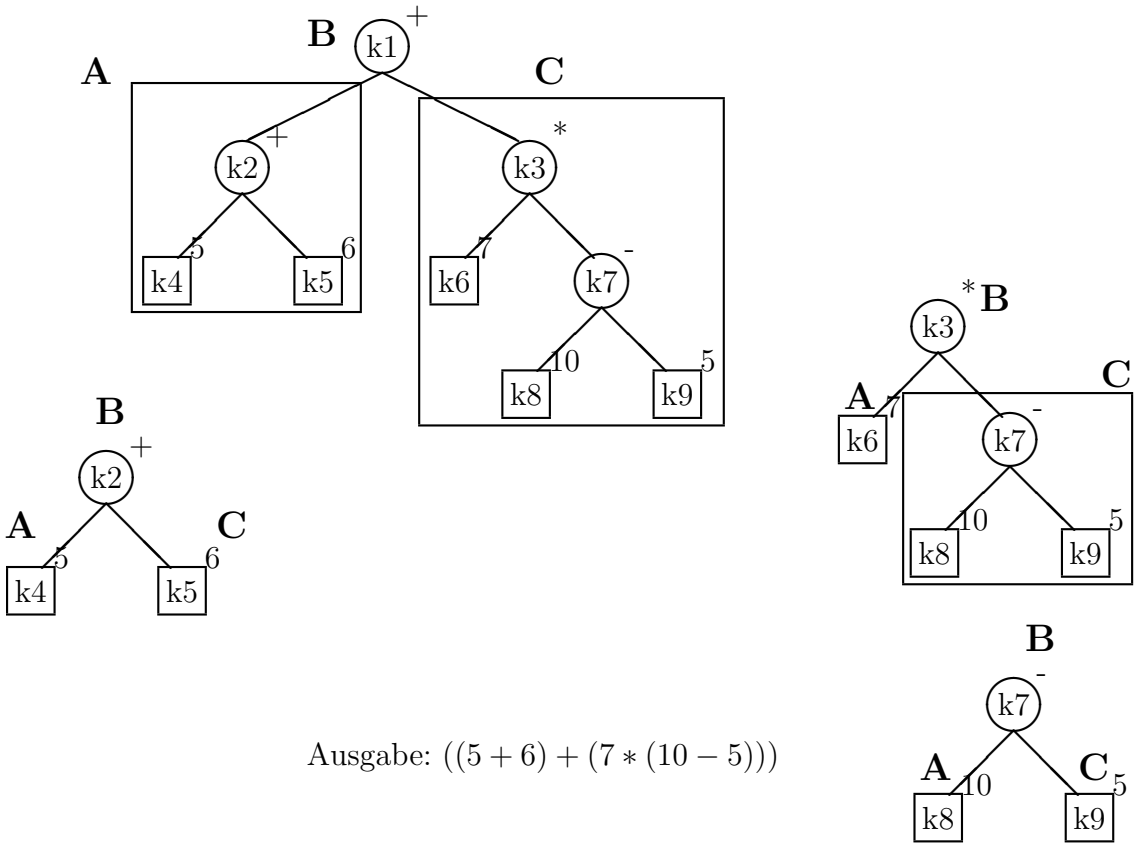
Präfix	*Infix*	*Postfix*
	$1 - 4 \div 2$	
	$1 \div 4 - 2$	
		2 3 4 ÷ -

Dies waren alles sehr einfache Ausdrücke. Es ist aber kein leichtes, die Ausdrücke von Hand in die anderen Notationen zu überführen. Stellen wir die Ausdrücke als Baum dar, kann die Aufgabe um einiges erleichtert werden.

Dazu müssen wir zuerst das richtige Werkzeug bereitstellen. Bei allen drei Notationen bleibt die Reihenfolge der Zahlen gleich. Was sich ändert, das ist die Anordnung der Operatoren. Bei den Operatoren kann sich sowohl die Position als auch die Reihenfolge ändern. Übertragen wir diese Beobachtung auf die Baumdarstellung, siehe Abb. 2.3. Die Zahlen werden alle in den Blättern abgebildet. Der Baum wird von links nach rechts traversiert. Dabei wird die Zahl bei jedem erreichten Blatt aufgeschrieben. Es ist zu beobachten, dass die Reihenfolge der aufgeschriebenen Zahlen mit der Reihenfolge der Zahlen in allen Notationen übereinstimmt. Vergleichen Sie Abb. 2.4.
Die entscheidende Frage ist: In welcher Reihenfolge müssen die inneren Knoten ausgegeben werden, so dass die Operatoren richtig positioniert werden?

Betrachten wir zuerst die Infix Ausdrücke. Für einen Ausdruck mit einem Operator und zwei Operanden steht bei dieser Notation links der erste Operand, dann in der Mitte der Operator und rechts steht der zweite Operand. Dies entspricht im Baum der Reihenfolge linkes Blatt, Knoten und dann rechtes Blatt. Wird der Ausdruck grösser, können wir dieses Schema erweitern. Damit die Infix-Notation eindeutig ist, muss sie geklammert werden. Dazu wird zuerst die öffnende Klammer gesetzt. Danach folgt der linke Teilbaum, Knoten und rechte Teilbaum. Zum Abschluss wird dann jewiel noch die schliessende Klammer gesetzt.

Links steht der erste Teilausdruck, dann der Operator und rechts der zweite Teilausdruck. Damit wir beim Baum die gleiche Reihenfolge erhalten, müssen wir die Knoten wie folgt traversieren. Zuerst linker Teilbaum, dann der Knoten und danach der rechte Teilbaum.



Ausgabe: $((5 + 6) + (7 * (10 - 5)))$

Abbildung 2.5: Inorder Traversierung für Infix Ausdrücke

Fassen wir die drei Schritte für die infix Traversierung nochmals zusammen.

Inorder-Traversierung

- Schritt A
  '(' Der linke Teilbaum wird traversiert. Falls dieser leer ist, mache nichts.
- Schritt B
  Der Knoten wird ausgegeben.
- Schritt C
  Der rechte Teilbaum wird traversiert. Falls dieser leer ist, mache nichts. ')'

Diese Art einen Baum zu traversieren wird Inorder-Traversierung genannt. Beim Traversieren des linken und rechten Teilbaumes wird wieder nach den drei Schritten verfahren.

Inorder-Traversierung Beispiel

Die Abb. 2.5 zeigt die Anwendung der drei Schritte der Inorder-Traversierung an einem Baum.

Die Traversierung beginnt bei der Wurzel mit dem Namen *k1*.
Klammer auf.
Nach Schritt A muss zuerst der linke Teilbaum traversiert werden.
    Links unten auf Abb. 2.5 ist der linke Teilbaum dargestellt.
    k2 ist die Wurzel des linken Teilbaumes, auf der nun die Traversierung beginnt.
    Klammer auf.
    Wieder nach Schritt A muss der linke Teilbaum zuerst traversiert werden.
        Jetzt startet die Traversierung auf *k4*.
        Hier kann die öffnende Klammer weggelassen werden.
            Schritt A macht jetzt nichts, da der linke Teilbaum von *k4* leer ist.
        Nach Schritt B muss der Knoten ausgegeben werden, dies ist jetzt *k4*.
        Beim Ausgeben schreiben wir einfach den Inhalt der Etikette auf.
        Also wie auf der Abb. 2.5 wird 5 zuerst aufgeschrieben.
            Schritt C ist auch einfach, da auch der rechte Teilbaum von *k4* leer ist.
            Hier muss die Klammer auch wieder weggelassen werden.
        Somit ist die Traversierung des Teilbaumes mit der Wurzel *k4* abgeschlossen.
    Das bedeutet, dass bei der Traversierung des Teilbaumes mit der Wurzel *k2* der linke Teilbaum erfolgreich traversiert ist und zu Schritt B übergegangen werden kann.
    Schritt B gibt jetzt den Knoten aus.
    Die Etikette vom Knoten *k2* wird aufgeschrieben.
        Nach Schritt C muss jetzt der rechte Teilbaum mit Wurzel *k5* traversiert werden.
        k5 ist identisch zu *k4*.
        6 wird aufgeschrieben und die Klammer geschlossen.
    Klammer zu.
Jetzt ist der ganze linke Teilbaum von *k1* traversiert.

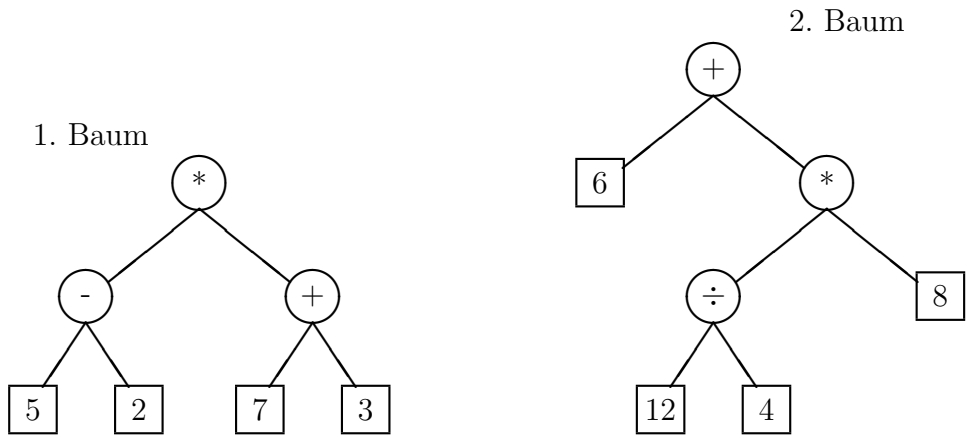


Abbildung 2.6: Die Bäume für die Aufgaben zu Vertiefung der Traversierung. Zur Vereinfachung der Darstellung verzichten wir auf die Namen der Knoten und schreiben gleich die Etiketten als Inhalt in die Knoten.

Nach Schritt B wird + aufgeschrieben.

Schritt C verlangt nun die Traversierung vom rechten Teilbaum von *k1*. Dieser Baum hat die Wurzel *k3*. Der Teilbaum mit Wurzel *k3* wird nach den gleichen Regeln abgearbeitet.

Ganz am Schluss machen wir wieder die Klammer zu.

Gehen Sie den Rest des Baums alleine durch. Falls Ihnen etwas unklar ist, dann betrachten Sie nochmals den linken Teilbaum. Vergessen Sie nicht die Klammern zu schliessen. Beim Traversieren wurde bis jetzt der traversierte Knoten ausgegeben. Anstatt den Knoten nur auszugeben, kann er auf eine beliebige Weise verarbeitet werden.

Aufgabe 2.6 *Vertiefung der Inorder-Traversierung. Stellen Sie die Ausdrücke, die in den Bäumen in Abbildung 2.6 repräsentiert sind, in der Infix-Notation dar. Schreiben Sie dazu die Knoten der Bäume in der Inorder Reihenfolge auf. Denken Sie daran, dass bei der Infix-Notation Klammern nötig sind, um den Ausdruck eindeutig zu machen.*

Damit wir die Präfix-Notation erhalten, muss nun die Reihenfolge der Operanden (der inneren Knoten) geändert werden. Wird bei dem Baum nun zuerst die Wurzel, dann der linke und danach der rechte Teilbaum traversiert, dann erhalten wir die gewünschte Reihenfolge. Dies ist die Preorder-Traversierung. Fassen wir die drei Schritte für die Preorder-

Traversierung nochmals zusammen.

Preorder-Traversierung

- Schritt A
  Der Knoten wird ausgegeben.
- Schritt B
  Der linke Teilbaum wird traversiert. Falls dieser leer ist, mache nichts.
- Schritt C
  Der rechte Teilbaum wird traversiert. Falls dieser leer ist, mache nichts.

Aufgabe 2.7 *Vertiefung der Preorder-Traversierung.*
Beschreiben Sie die Anwendung der Preorder-Traversierung auf dem dem 1. Baum in Abb. 2.6, wie es in bei der Inorder-Traversierung geben ist. Stellen Sie danach die Ausdrücke die in den Bäumen in der Abbildung 2.6 repräsentiert sind, in der Präfix-Notation dar. Schreiben Sie dazu die Knoten der Bäume in der Preorder Reihenfolge auf.

Die Postfix-Notation erhält man mit einer weiteren Traversierungsart. Bei der Postfix-Notation steht der Operator nach den beiden Operanden. So muss der Knoten auch nach den beiden Teilbäumen traversiert werden. Also zuerst linker Teilbaum, dann der rechte Teilbaum und danach der Knoten. Eine solche Traversierung wird als Postorder-Traversierung bezeichnet.

Postorder-Traversierung

- Schritt A
  Der rechte Teilbaum wird traversiert. Falls dieser leer ist, mache nichts.
- Schritt B
  Der linke Teilbaum wird traversiert. Falls dieser leer ist, mache nichts.
- Schritt C
  Der Knoten wird ausgegeben.

Aufgabe 2.8 *Vertiefung der Postorder-Traversierung. Stellen Sie die Ausdrücke, die in den Bäumen in Abbildung 2.6 repräsentiert sind in der PostfixNotation dar. Schreiben Sie dazu die Knoten der Bäume in der Postorder Reihenfolge auf. Falls Sie etwas unsicher sind, schreiben Sie auch hier die Anwendung der Traversierung auf.*

Um die mathematischen Ausdrücke in Bäume zu überführen, muss man die Traversierungen umgekehrt anwenden. Den Baum aufbauen, anstatt ihn auszulesen. Jetzt haben wir alle Werkzeuge beisammen, um die mathematischen Ausdrücke in die verschiedenen Notationen zu überführen.

Aufgabe 2.9 *Hier sind nochmals mathematische Ausdrücke, die in die jeweils beiden anderen Notationen übersetzt werden sollen. Versuchen Sie dieses Mal zuerst den Baum aufzuzeichnen, dann können Sie mit den gelernten Traversierungen nur noch die anderen zwei Schreibweisen ablesen. Dieses Mal sollten sie die Infix-Ausdrücke mit Klammern versehen, da sie nicht eindeutig sein müssen ohne Klammern.*

Präfix	*Infix*	*Postfix*
$+\ 6 * 7 \div 6\ 2$		
	$(((3+1)*4) \div (10-6))$	
		$2\ 10 + 9 - 8\ 3 + *$

Mit der Hilfe von Bäumen ist die Umwandlung von mathematischen Ausdrücken zu einem einfachen Problem geworden. Programmcode lässt sich ähnlich wie mathematische Ausdrücke repräsentieren. Die einzelnen Statements (Befehle) lassen sich als Bäume darstellen. Diese Bäume werden dann in einer Liste zusammengefügt. Eine Liste von Bäumen wird Wald genannt. Der Wald beschreibt das ganze Programm.

2.6 Zusammenfassung

Fassen wir die erarbeiteten Anforderungen zusammen.

- Wir möchten maximal 2 Kinder pro Knoten, falls das möglich ist.
  Das vereinfacht die Struktur.
- Wir möchten eine Ordnung bei den Kindern.
  Wir können zwischen linkem und rechtem Kind entscheiden.
- Es gibt drei mögliche Traversierungsarten
  - Preorder-Traversierung
    für die Präfix-Notation
  - Inorder-Traversierung
    für die Infix-Notation
  - Postorder-Traversieurng
    für die Postfix-Notation
- Die Struktur des Baumes ist stabil, das heisst, sie bleibt über längere Zeit unverändert.
  Ein Ausdruck oder der Aufbau einer Organisation wird sich nicht verändern, das heisst auch der Baum bleibt immer gleich.

2.7 Lösungen zu den Aufgaben

Antwort 2.1 *Überprüfen Sie folgende Eigenschaften an ihrem Baum:*

- *Hat ihr Baum eine Wurzel*
- *Hat jeder Knoten nur einen Vater*
- *Stimmt die Struktur des Baumes mit der Gliederung des Inhaltsverzeichnis überein*

Wenn all diese Punkte erfüllt sind, wird ihr Baum korrekt sein.

Antwort 2.2 *Hier gibt es keine Lösung.*

Antwort 2.3 *Wir unterteilen den Summen-Operator auf zwei Knoten. Einen Grenzknoten und einen Summenknoten. Der Grenzknoten hat als Kinder die untere und obere Grenze, die Blätter (Zahlen, Konstanten) sind. Der Summenknoten hat jetzt auch nur noch zwei Kinder. Den Grenzknoten und den Ausdrucksknoten.*

Antwort 2.4 *Durch die Anordnung der Kinder kann gezeigt werden welches Kind jünger ist. Die Kinder müssen dazu von links nach rechts im Alter auf- oder absteigend angeordnet werden. Es ist wichtig dass es in einem Baum immer gleich gemacht wird und dass klar gesagt wird ob die Kinder auf- oder absteigend angeordnet sind.*

Antwort 2.5 *Die vervollständigte Liste der Ausdrücke.*

Präfix	*Infix*	*Postfix*
- 1 ÷ 4 2	1-4÷2	1 4 2 ÷ -
- ÷ 1 4 2	1÷4-2	1 4 ÷ 2 -
- 2 ÷ 3 4	2 - 3 ÷ 4	2 3 4 ÷ -

Es wurden absichtlich nicht kommutative Operatoren gewählt, damit die Lösungen eindeutig sind. Die Infix-Notation könnte man für bessere Lesbarkeit klammern. Sie ist aber auch ohne Klammerung eindeutig, da die Bindungstärke der Operatoren festgelegt ist. Bei den Postfix- und Präfix-Notationen ist eine Klammerung nicht notwendig, da die Darstellung eindeutig ist.

Antwort 2.6 *Die Inorder-Traversierung führt zu folgenden Ausdrücken:*
*1. Baum: ((5 - 2) * (7 + 3))*
*2. Baum: (6 + ((12 ÷ 4) * 8))*

Antwort 2.7 *Achten Sie darauf, dass Sie jeden Schritt immer zuendebringen, bevor Sie zum nächsten Schritt gehen. Überprüfen Sie ihre Anleitung anhand der drei Schritte der Preorder-Traversierung.*
Die Preorder-Traversierung führt zu folgenden Ausdrücken:
*1. Baum: * - 5 2 + 7 3*
*2. Baum: + 6 * ÷ 12 4 8*

Antwort 2.8 *Die PostorderTraversierung führt zu folgenden Ausdrücken:*
*1. Baum: 5 2 - 7 3 + **
*2. Baum: 6 12 4 ÷ 8 * +*

Antwort 2.9 *Die vervollständigte Liste der Ausdrücke.*

Präfix	*Infix*	*Postfix*
$+\ 6 * 7 \div 6\ 2$	$(6 + (7 * (6 \div 2)))$	$6\ 7\ 6\ 2 \div * +$
$\div * +3\ 1\ 4 - 10\ 6$	$(((3 + 1) * 4) \div (10 - 6))$	$3\ 1 + 4 * 10\ 6 - \div$
$* - +2\ 10\ 9 + 8\ 3$	$(((2 + 10) - 9) * (8 + 3))$	$2\ 10 + 9 - 8\ 3 + *$

Kapitel 3

Implementation Etiketten und AusdrucksBäume

3.1 Übersicht

Etikettenbäume werden verwendet, um hierarchische Daten zu verwalten. Die Anordnung der Knoten in den Bäumen ist ein Teil der Information. Eine Implementation eines Etikettenbaums kann oft nur für wenig Anwendungen verwendet werden. In diesem Kapitel erarbeiten Sie zwei Implementationen. Die eine für die Abbildung eines Organisationsdiagrammes und die andere zur Speicherung mathematischer Ausdrücke.

3.1.1 Was lernen Sie hier?

Sie lernen, wie man einen Etikettenbaum für gegebene Anforderungen erstellt. Sie sehen, wie ein Baum aus den Knoten aufgebaut wird. Sie implementieren die drei Traversierungsarten und lassen den Computer die verschiedenen Notationen erarbeiten.

3.2 Lernziele

Nachdem Sie dieses Kapitel durchgelesen haben,

wissen Sie, weshalb Etikettenbäume oft für jede Anwendung neu implementiert werden müssen.

kennen Sie ein Schema, mit dem Sie gezielt Code generieren können.

können Sie für gegebene Anforderung eine Implementation eines Etikettenbaumes erstellen.

können Sie die drei Traversierungen rekursiv implementieren.

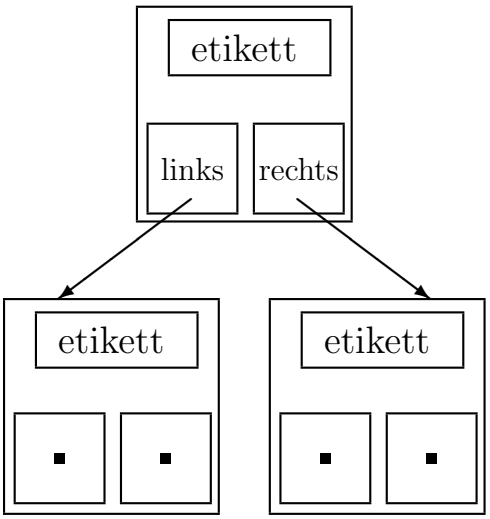


Abbildung 3.1: Design: Drei Knoten, die miteinander als Baum verbunden sind.

3.3 Das Design

Wir wählen hier eine einfache Struktur. Die Bäume werden so implementiert, wie wir sie aufgezeichnet haben. Jeder Knoten wird durch ein Knoten-Objekt repräsentiert. Ein Knoten verweist auf all seine Kinder. Die Abb. 3.1 zeigt drei Knoten die einen Baum bilden. Falls es erforderlich ist, dann kann ein Knoten auch auf den Vater verweisen.

Aufgabe 3.1 *Verweise zum Vater. Erweitern Sie die Darstellung in Abb. 3.1 um ein Feld in jedem Knoten, welches auf den Vater verweist. Vom neuen Feld soll dann ein Pfeil auf den Vaterknoten zeigen. Zeichnen Sie die ganze Abbildung neu.*

Die Struktur soll sehr offen sein. So kann der Baum vom Anwender aufgebaut werden. Dies ist notwendig, da der Baum nicht selbständig neue Knoten einfügen kann, wenn diese eine vorbestimmte Position haben, die der Baum nicht kennt. Dieser offene Ansatz ist gefährlich. Es ist wichtig, dass ein Anwender nichts machen kann, was den Baum zerstört. Ein Baum ist nicht zerstört, wenn alle Knoten gelöscht werden. Der Baum ist zerstört wenn er in einer Konfiguration ist, die nicht zulässig ist. Verweist ein Knoten auf ein Kind und dieses Kind verweist auf einen anderen Vater, dann haben wir eine unzulässige Konfiguration.

3.4 Baum für mathematische Ausdrücke

Die Anforderungen an einen Etikettenbaum zur Repräsentation von mathematischen Ausdrücken sind am Ende des Kapitel 2 aufgelistet. Die Herausforderung besteht darin eine Datenstruktur zu entwerfen, die all diesen Anforderungen gerecht wird. Um dieses Ziel zu erreichen werden wir wie folgt vorgehen.

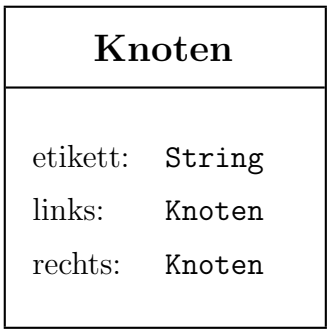


Abbildung 3.2: Der Knoten

Zuerst wird erarbeitet, welche Daten auf die Etikette kommen. Danach betrachten wir den Aufbau des Knotens. Dort wird bestimmt, welche Verweise in den Knoten kommen. Anhand der Operationen, die der Baum ausführen sollte, wird ein Interface für den Knoten erarbeitet. Sind diese Schritte durchgeführt, wird die Implementierung des Knotens ausgearbeitet.

3.4.1 Bestimmen der Etikette

Die mathematischen Ausdrücke bestehen aus den Operatoren und den Operanden. Es gibt eine endliche Liste der Operatoren. Es muss klar definiert werden, welche Operatoren vom Baum unterstützt werden, damit sie richtig abgebildet werden können. Als Operanden gibt es Zahlen und Variablen. Die Zahlen werden am einfachsten in `int` Variablen abgespeichert. Die Variablen hingegen müssen als `String` gespeichert werden.
Dies könnte durch eine Klasse mit verschiedenen Feldern erreicht werden. Wir machen an dieser Stelle eine Vereinfachung, indem wir die Zahlen auch als Strings darstellen. Die Operatoren, Variablen und Zahlen werden als Text abgespeichert. Diese Vereinfachung macht hier nur Sinn, da es in erster Linie um die Repräsentation der Ausdrücke geht. Steht das Arbeiten mit den Zahlen im Vordergrund würden umgekehrt die Operatoren auf Zahlen abgebildet oder eine Klasse mit mehreren Felder eingesetzt.

Das Etikett ist für unseren MathAusdrucksBaum vom Typ `String`.

3.4.2 Aufbau eines Knotens

Welche Felder werden vom Knoten benötigt, damit ein Baum aufgebaut werden kann, der die Anforderungen erfüllt? Die geordneten zwei Kinder und die Etikette sind essentielle Bestandteile eines jeden Knotens. Abb. 3.2 zeigt die Klasse Knoten mit den minimal erforderlichen Feldern. Soll der Knoten ein Blatt darstellen, so hat er keine Kinder. Wenn der Knoten keine Kinder hat, werden die Felder für die Kinder einfach auf `null` gesetzt.

Aufgabe 3.2 *Soll der Baum auch Verweise auf den Vater beinhalten, dann benötigen wir weitere Felder. Zeichenen Sie die Knotenklasse mit der auch auf den Vater verwiesen werden kann.*

Der folgende Javacode implementiert den minimalen Knoten.

```
class MathAusdrKnoten
{
  // Felder
  /** Das linke Kind */
  private MathAusdrKnoten _links = null;
  /** Das rechte Kind */
  private MathAusdrKnoten _rechts = null;

  /** Etikett; Inhalt des Knoten */
  private String _zeichen = null;

  // ...

}
```

Auf der Basis von der Klasse `MathAusdrKnoten` können die Bäume aufgebaut werden.

Aufgabe 3.3 *Die Klasse `MathAusdrKnoten` implementiert den minimalen Knoten. Schreiben Sie die noch benötigten Felddeklarationen, damit auch auf den Vater verwiesen werden kann.*

3.4.3 Interface eines Knotens

Welche Operationen soll der Knoten anbieten? Der Zugriff auf die Kinder soll möglich sein. Die Kinder sollen gelesen und geschrieben werden dürfen. Das Schreiben ist notwendig, um den Baum aufzubauen. Das Lesen wird benötigt, um den Baum von aussen zu traversieren. Methoden für den Zugriff auf die Kinder.

```
  // Zugriff auf die Felder
  /** Gibt das linke Kind zurück */
  public MathAusdrKnoten links() { ... }

  /** Setzt das linke Kind. */
  public void setLinks(MathAusdrKnoten links) { ... }

  /** Gibt das rechte Kind zurück */
  public MathAusdrKnoten rechts() { ... }

  /** Setzt das rechte Kind. */
  public void setRechts(MathAusdrKnoten rechts) { ... }
```

Mit diesen Methoden ist es jederzeit möglich, die Struktur des Baumes zu ändern. Der Anwender muss achten, dass keine ungewollten Änderungen stattfinden.

Die Etikette soll auch gelesen und beschrieben werden können.

```
/** Gibt das/die im Knoten gespeicherte Zeichen/Zahl zurück. */
public String zeichen() { ... }

/** Überschreibt die Etikette mit dem neuen Inhalt */
public void setZeichen(String z) { ... }
```

Zu den einfachen Feldzugriffen auf den Knoten, soll der aus mehreren Knoten aufgebaute Baum noch kompliziertere Operationen anbieten. Zum Beispiel die Traversierung des Baumes. Unser Baum besteht nur aus einzelnen Knoten. Jeder Knoten soll die Traversierung des Baumes anbieten, von dem der Knoten die Wurzel ist. Besteht also ein Baum aus vielen Knoten, so muss die Traversierung auf der Wurzel gestartet werden, damit der ganze Baum traversiert wird. Jeder Knoten soll Methoden anbieten, die den zugehörigen Baum in Post- Pre- und In-Order traversiert und die Etiketten der Reihe nach ausgibt.

```
// Traversierungen
/** Gibt den Teilbaum, der diesen Knoten als Wurzel hat,
 * in der In-Order Reihenfolge auf die Konsole aus.
 */
public void printInorder() { ... }

/** Gibt den Teilbaum, der diesen Knoten als Wurzel hat,
 * in der Pre-Order Reihenfolge auf die Konsole aus.
 */
public void printPreorder() { ... }

/** Gibt den Teilbaum, der diesen Knoten als Wurzel hat,
 * in der Post-Order Reihenfolge auf die Konsole aus.
 */
public void printPostorder() { ... }
```

Mit diesen drei Methoden lassen sich, die als Baum dargestellten mathematischen Ausdrücke spielerisch in alle drei Notationen umwandeln.

Zum Schluss soll der Knoten auch noch eine sinnvolle Methode anbieten. Die Methode `auswerten()` soll den zum Knoten gehörenden Baum auswerten.

```
/** Berechnet den mathematischen Ausdruck.
 * Um die Struktur einfach zu halten, werden nur die vier
 * Grundoperatoren +, -, *, / unterstützt.
 * Sind andere Operatoren im Baum oder ist der Baum nicht
 * korrekt, ist das Verhalten unbestimmt.
 */
public int auswerten() { ... }
```

Wie im Javadoc-Code geschrieben ist, implementieren wir hier nur eine einfache Auswertungsmethode. Sie auszubauen ist nicht kompliziert, macht aber in diesem Beispiel keinen Sinn.

Aufgabe 3.4 *Alle notwendigen Interfaces wurden bis jetzt gezeigt. Sie wollen aber wieder einen Verweis auf den Vater Knoten in ihrem Baum haben. Welche zusätzlichen Interfaces sind dafür notwendig. Schreiben Sie diese Interfaces auf.*

Damit die Klasse instanziert werden kann, braucht es auch noch einen Konstruktor. Der Konstruktor soll als Parameter ein `String` nehmen und die Etikette initialisieren.

3.4.4 Implementation

Das Interface der Klasse `MathAusdrKnoten` ist festgelegt. Auch sind die drei wichtigsten Felder deklariert. Jetzt müssen nur noch die Methoden ausprogrammiert werden. Wir werden gemeinsam den Code der Preorder-Traversierung betrachten. Die Inorder- und Postorder-Traversierungen können Sie danach selbst implementieren.

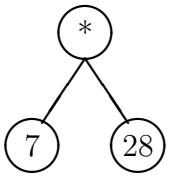
Der Code für die Preorder-Traversierung funktioniert genau nach dem Prinzip das in Kapitel 2 vorgestellt wurde. Der Knoten gibt zuerst seine Etikette aus (Zeile 5). Das Leerzeichen wird ausgegeben, damit die einzelnen Elemente des Ausdruckes auseinander gehalten werden können. Nach der Ausgabe des Knoten muss zuerst der linke und dann der rechte Teilbaum ausgegeben werden.

```
1  /** Gibt den Teilbaum, der diesen Knoten als Wurzel hat,
2   * in der Pre-Order Reihenfolge auf die Konsole aus.
3   */
4  public void printPreorder() {
5    System.out.print(_zeichen + " ");                  // Knoten ausgeben
6    if(_links != null) { _links.printPreorder(); }     // Linker Teilbaum
7    if(_rechts != null) { _rechts.printPreorder(); }  // Rechter Teilbaum
8  }
```

Falls ein Verweis auf ein Kind `null` ist, dann hat der Knoten auf dieser Seite kein Kind mehr. Auf Zeile 6 wird geschaut, ob ein linker Teilbaum vorhanden ist. Hat der Knoten ein linkes Kind, dann wird auf dem linken Kind `printPreorder()` aufgerufen. Dieser Aufruf bewirkt dass das Kind (linker Teilbaum) traversiert wird. Ist das linke Kind nicht vorhanden, wird nichts gemacht. Jetzt muss noch der rechte Teilbaum traversiert werden. Hier werden die gleichen Überprüfungen wie beim linken Teilbaum gemacht.

Aufgabe 3.5 *Im bereitgestellten Code Ordner befindet sich die Datei* Mathausdrknoten.java. *Suchen Sie sich einen Partner und vervollständigen Sie zusammen mit ihm die Inorder- und Postorder-Traversierungen. Die Stellen sind mit einem* `/*TODO*/` *gekennzeichnet. Kompilieren und testen Sie ihren Code. Für den Test können Sie die das Programm in der* Test1.java *Datei verwenden. Die erwartete Ausgabe ist in der Datei als Kommentar vorhanden.*
Tipp: Die Infix-Notation ist ohne Klammerung nicht eindeutig. Fügen Sie bei der Inorder-Traversierung die Klammern ein. Klammern Sie der Einfachheit halber den Ausdruck vollständig.

Hier können Sie weiter in der Zweiergruppe arbeiten.
In der Datei *Test1.java* können Sie sich anschauen, wie der Baum aufgebaut wird. Die einzelnen Knoten werden zu einem immer grösser werdenden Baum zusammengehängt.




```
MathExprKnoten k1,k2,k3;
k1 = new MathExprKnoten("7");
k2 = new MathExprKnoten("*");
k2.setLinks(k1);
k1 = new MathExprKnoten("28");
k2.setRechts(k1);
```

Der obige Teilbaum wurde aus dem Code rechts davon erstellt. Schauen Sie sich an, wie der Baum aufgebaut wird.

Aufgabe 3.6 *Diese Aufgabe soll auch wieder im Zweierteam bearbeitet werden. Jeder von Ihnen soll sich zwei mathematische Ausdrücke überlegen. Achten Sie darauf, dass Sie nur die 4 Operatoren +, -, $*$ und $\div$ verwenden und keine Variablen in Ihren Ausdrücken vorkommen. Diese vier Ausdrücke sollen Sie dann zusammen in die* Test1.java *Datei einfügen. Bauen Sie die vier zugehörigen Bäume und geben Sie diese in allen drei Notationen aus. Speichern Sie die Datei, diese Bäume werden in einer späteren Aufgabe weiter verwendet.*

Die Methode `auswerten()` ist noch nicht implementiert. Die Methode soll als eine Traversierung des Baumes implementiert werden. Aber anstatt die Knoten auf die Konsole zu schreiben wird der Knoten ausgewertet. Die Methode hat die Signatur `int auswerten()`. Ist der Knoten ein Blatt, dann soll die Methode den Wert zurück geben, der im Knoten gespeichert ist. Dafür kann der Befehl `int Integer.parseInt(String)` verwendet werden. Ist der Knoten kein Blatt, dann muss im Knoten ein Operator sein. Dies kann mit der `equals(String)` Methode aus der Klasse `String` überprüft werden. Bei Unklarheiten schauen Sie in der Java API Dokumentation [7] nach. Ist der Operator erkannt, dann kann der Teilausdruck in diesem Knoten berechnet werden und das Resultat wird zurück gegeben, damit der Vaterknoten seinen Teilausdruck auch berechnen kann.

Aufgabe 3.7 *Implementieren Sie die* `auswerten()` *Methode zusammen mit einem Partner. Beschränken Sie sich auf die vier Grundoperatoren. Testen Sie ihre Implementation mit dem in der Testdatei gegebenen Baum. Dazu müssen Sie bei der markierten Stelle nur das Kommentarzeichen entfernen.*
Ist die Berechnung erfolgreich verlaufen? Werten Sie nun ihre vier eigenen Bäume aus.

Gratulation. Sie haben einen Baum zur Repräsentation von mathematischen Ausdrücken implementiert und diesen auch angewendet. Solche Bäume werden in vielen modernen Programmen verwendet. Das Programm *javac* verwendet solche Bäume zur Darstellung des Java-Quellcodes. Auf dem so dargestellten Code führt das Programm zahlreiche Optimierungen durch, damit ihr Programm optimal abläuft.

3.5 Baum für Organisationsdiagramm

Die Anforderungen für einen Baum zur Darstellung von Organisationsdiagrammen haben wir auch in Kapitel 2 erarbeitet. Die Anforderungen unterscheiden sich etwas von denen für die mathematischen Ausdrücke. Die Traversierungen haben wir bisher nur für binäre Bäume besprochen. Bei Bäumen höherer Ordnung ist die Inorder-Traversierung nicht mehr definiert. Die beiden anderen unterscheiden sich in der Definition nicht.

Aufgabe 3.8 *Schreiben Sie eine Knoten Klasse, aus der Bäume zur Darstellung von Organisationsdiagrammen erstellt werden können. Gehen Sie mit einem Partner die gleichen Schritte wie in Abschnitt 3.4 durch.*
Verwenden Sie als Etikette diesmal eine Klasse `Person`*. Diese Klasse sollte alle notwendigen Eigenschaften einer Person in diesem Kontext enthalten. Überlegen Sie sich, wie Sie einfach die Frage nach dem Vorgesetzten beantworten können. Als weiterer Punkt müssen Sie einen Weg finden, beliebig viele Kinder zu ermöglichen. Verwenden Sie hierzu wenn möglich Strukturen die Ihnen vom Java API zur Verfügung gestellt werden.*
Die Traversierungen müssen Sie nicht noch einmal implementieren. Falls Sie ein Problem haben, bitten Sie die betreuende Person um Hilfe oder werfen Sie einen kurzen Blick in die Lösung.

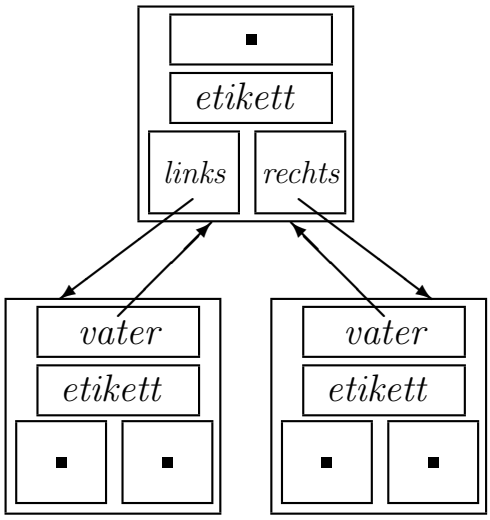
3.6 Schlussbemerkung

Sie haben jetzt zwei Bäume programmiert. Die zweite Implementierung ist sehr allgemein. Mit dem ersten Baum können gerade mal mathematische Ausdrücke repräsentiert werden, viel mehr kann man damit nicht anfangen. Die Implementation des Baumes für das Organisationsdiagramm kann noch verallgemeinert werden, indem der Typ beim Feld zur Speicherung des Etiketts durch `Object` ersetzt wird. Nun könnten alle baumartigen Strukturen auf diesen Baum abgebildet werden, da als Etikette jedes Objekt akzeptiert wird. Leider ist diese allgemeine Struktur selten ideal für eine speziele Struktur geeignet. Wie Sie gesehen haben, sind die Etikettenbäume sehr einfach. Die Bäume sind mit der Anwendung

sehr eng verbunden. Die enge Angliederung an eine Anwendung und die Einfachheit dieser Bäume bewirken, dass Etiketten-Bäume praktisch bei jeder Anwendung neu implementiert werden.

3.7 Lösungen zu den Aufgaben

Antwort 3.1 *Die folgende Abbildung zeigt die gleiche Situation mit Verweisen auf den Vaterknoten.*



Antwort 3.2 *Das Feld Vater wird zusätzlich eingefügt. Es ist vom Typ* `Knoten`.

Knoten	
vater:	`Knoten`
etikett:	`String`
links:	`Knoten`
rechts:	`Knoten`

Antwort 3.3 *Damit auf den Vater verwiesen werden kann, wird noch ein Feld* `vater` *benötigt.*

```
//...
// zusätzliches Feld
/** Der Vater */
private MathExprKnoten _vater = null;
// ...
```

Antwort 3.4 *Es werden noch die Interfaces für das Setzen und Abfragen des Vaters benötigt. Das Setzen des Vaters könnte weggelassen werden, wenn ein explizites setzten nicht verlangt ist. Da die Implementation der Methoden zum setzten der Kinder auch die Vaterbeziehung setzten kann. Dies braucht keine Anpassung des Interfaces und garantiert, dass die Verweise auf den Vater immer korrekt sind.*

```
// Zugriff auf die Felder
//...

/** Gibt den Vater zurück */
public MathAusdrKnoten vater() { ... }

/** Setzt den Vater. */
public void setVater(MathAusdrKnoten vater) { ... }
```

Antwort 3.5 *War die Ausgabe korrekt? Falls nicht korrigieren Sie ihr Programm. Eine Beispiellösung finden Sie im Ordner* Loesungen.

Antwort 3.6 *Hier kann keine Lösung für Ihre Bäume stehen, da diese mir nicht bekannt sind. Vergleichen Sie die drei erhaltenen Notationen mit Ihren ursprünglichen Ausdrücken. Stimmen sie überein, dann haben Sie gut gearbeit*

Antwort 3.7 *Eine Beispiellösung finden Sie im Ordner* Loesungen.

Antwort 3.8 *Im Ordner* Loesungen *finden Sie ein Beispielprogramm. Vielleicht haben Sie eine bessere Lösung. Dieser Code ist nur ein Vorschlag, der die Anforderungen zu erfüllen scheint.*

Kapitel 4

Konzept Suchbäume

4.1 Übersicht

4.1.1 Was lernen Sie hier?

Im Kapitel 2 wurden Bäume für die Speicherung und Verarbeitung von hierarchischen Strukturen diskutiert. In diesem Kapitel wird die Baumstruktur eingesetzt, um eine einfache Handhabung einer Menge zu erreichen.
Wir diskutieren die Eigenschaften einer Menge und werden versuchen diese Eigenschaften mit Hilfe der Bäume zu modellieren.

4.2 Lernziele

Nachdem Sie dieses Kapitel durchgelesen haben,

verstehen Sie, warum Bäume eine grosse Hilfe sind beim Verwalten von Mengen.
kennen Sie die drei Grundoperationen auf Mengen.
wissen Sie, wie sie funktionieren.
wissen Sie, was ein Suchbaum ist.

4.3 Suchbäume

Suchbäume sind geordnete binäre Schlüsselbäume. In Schlüsselbäumen besitzt jeder Knoten einen Schlüssel. Ein Schlüssel ist ein Element aus einer geordneten Menge. Wir beschränken uns hier auf Schlüsselbäume, bei denen jeder Schlüssel nur einmal in einem Baum vorkommen darf. Für die meisten Anwendungen ist diese Einschränkung nicht störend. Da die Knoten eindeutig mit einem Schlüssel bestimmt werden können, brauchen wir hier nicht zusätzlich noch einen Knotennamen.

Die Suchbäume sind die am meisten verwendete Art von Bäumen. Sie sind besonders gut für die Verwaltung von geordneten Mengen geeignet. Insbesondere wenn die Grösse und die Zusammensetzung der Menge sich häufig ändert. Die Baumstruktur ist beim Suchen wesentlich schneller als zum Beispiel verlinkte Listen. Falls die Menge sich nicht so häufig verändert, gibt es allerdings andere gute Lösungen.

4.3.1 Der Set

Ein Beispiel für die Verwaltung einer Menge ist ein *Set*. Set ist englisch und steht für Menge. Wir verwenden das Wort Set als ein abstrakter Datentyp (ADT). Ein abstrakter Datentyp ist eine mathematische Struktur mit klar definierten Eigenschaften. Der Set hat folgende Eigenschaften. Jedes Element in einem Set ist eindeutig, das heisst es kann kein Element zweimal im Set vorhanden sein. Ein Set kann alle Elemente einer bestimmten Menge enthalten. Zum Beispiel alle ganzen Zahlen oder die Adressen aller Personen in der Schweiz. Ein Set kann beliebig viel Elemente enthalten. Es ist immer möglich, Elemente hinzuzufügen, solange die Menge der Elemente nicht ausgeschöpft ist. Ist ein Set nicht leer, dann kann ein Element nach Wahl entfernt werden.

Damit die Elemente unterscheidbar sind, hat jedes Element einen eindeutigen Schlüssel. Bei den natürlichen Zahlen kann gerade die Zahl selbst als Schlüssel dienen. Bei den Adressen der Personen könnte deren AHV-Nummer als Schlüssel verwendet werden. Die Elemente werden über diesen Schlüssel identifiziert. Bei gegebener AHV-Nummer soll der Set die Adresse der Person finden, falls diese im Set enthalten ist.

Die drei beschriebenen Operationen *Einfügen*, *Suchen* und *Entfernen* heissen *Wörterbuchoperationen*. In einem Suchbaum lassen sich die Wörterbuchoperationen am einfachsten erreichen.

Aufgabe 4.1 *Können Sie erklären, weshalb die Wörterbuchoperationen so wichtig in der Informatik sind. In der Informatik müssen sehr oft Informationen (Daten) verwaltet werden.*

4.3.2 Die Ordnung des Suchbaumes

Die Stärke des Suchbaums ist die Ordnung unter den Knoten. Anders als bei den Etikettenbäumen muss im Suchbaum nicht eine fixe Struktur repräsentiert werden. Die genaue

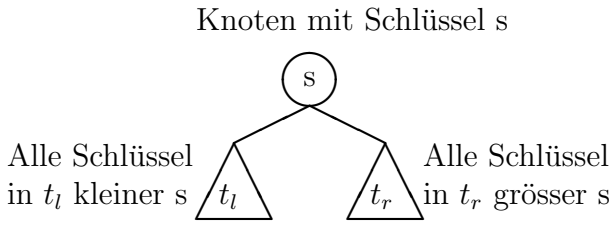
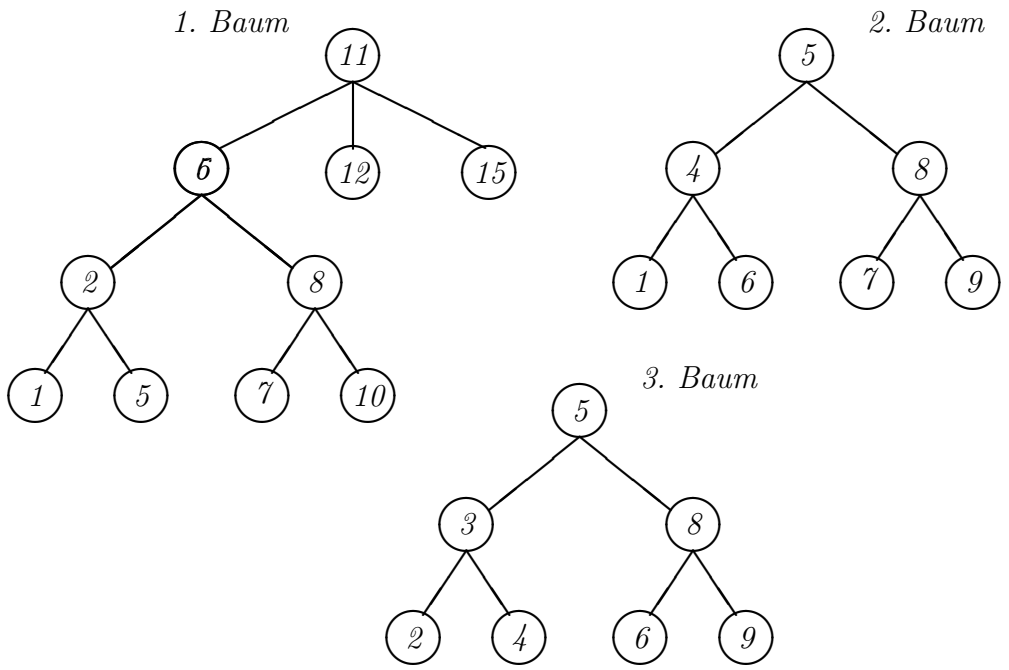


Abbildung 4.1: Die Ordnung der Schlüssel im Suchbaum.

Position eines bestimmten Knoten ist nicht wichtig. Wichtig ist, dass zu einem gegebenen Schlüssel der zugehörige Knoten rasch gefunden wird. Die in Abschnitt 4.3 verlangte Ordnung der Schlüsselmenge spielt hier eine entscheidende Rolle.

Der Suchbaum ist ein Binärbaum in dem alle Elemente im linken Teilbaum eines Knoten einen kleineren Schlüssel haben als der Knoten selbst hat. Im rechten Teilbaum sind alle Knoten mit einem grösseren Schlüssel als der des Knotens. Die Abb. 4.1 zeigt die Ordnung im Suchbaum.

Aufgabe 4.2 *Entscheiden Sie bei den folgenden Bäumen, welche von ihnen korrekte Suchbäume sind. Falls es Bäume gibt, die keine Suchbäume sind, dann bringen Sie diese in eine korrekte Form.*



Wie Sie gesehen haben, haben die Suchbäume eine ganz bestimmte Anordnung der Knoten. Diese Anordnung muss bei jeder Operation auf dem Baum erhalten bleiben.

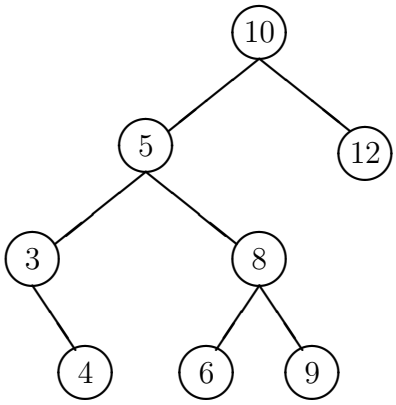


Abbildung 4.2: Ein Suchbaum.

Aufgabe 4.3 *Bei den Etikettenbäumen in Abschnitt 2.5 wurden verschiedene Traversierungsarten vorgestellt. Jede Traversierungsart brachte eine andere Notation der mathematischen Ausdrücke hervor. Wie muss ein Suchbaum traversiert werden, so dass die Schlüssel in aufsteigender Reihenfolge ausgegeben werden?*

4.3.3 Suchen im Suchbaum

Ist ein bestimmtes Element im Set enthalten? Diese Frage muss einfach beantwortet werden können. In Abb. 4.2 ist ein Suchbaum dargestellt. Wir nutzen die Ordnung des Baumes zur Suche eines bestimmten Schlüssels. Die Suche beginnt bei der Wurzel. Hat die Wurzel den gesuchten Schlüssel, dann ist die Suche fertig. Sonst gibt es zwei Fälle. Ist der zu suchende Schlüssel kleiner wie der in der Wurzel, dann geht die Suche im linken Teilbaum weiter. Ist der zu suchende Schlüssel grösser, dann geht es im rechten Teilbaum weiter. Ist der Schlüssel nicht im Baum, dann wird die Suche durch einen leeren Baum beendet. Dies trifft ein, falls die Suche in einem Blatt ist und die Schlüssel nicht übereinstimmen. Hier sollte die Suche in einem Teilbaum weiter gehen. Jetzt wird die Suche aber beendet, da der Teilbaum leer ist.

Suchen Beispiele

Suchen wir zusammen die Schlüssel 11 und 6 im Suchbaum in Abb. 4.2.
Suche nach dem Schlüssel 6 beginnt auf dem Baum mit Wurzel 10.
6 ist kleiner 10 → Suche geht im linken Teilbaum der an Knoten 5 wurzelt weiter.
6 ist grösser 5 → Suche geht im rechten Teilbaum der an Knoten 8 wurzelt weiter.
6 ist kleiner 8 → Suche geht im linken Teilbaum der an Knoten 6 wurzelt weiter.
6 entspricht 6 → Schlüssel gefunden, Suche ist beendet.

Suche nach Schlüssel 11 beginnt auf dem Baum mit Wurzel 10.
11 ist grösser 10 → Suche geht im rechten Teilbaum der an Knoten 12 wurzelt weiter.
11 ist kleiner 12 → Suche geht im linken Teilbaum weiter.
Leerer Baum → Suche beendet, Schlüssel nicht gefunden.

Aufgabe 4.4 *Das Suchen ist Bestandteil der nächsten Abschnitte. Suchen Sie noch nach dem Schlüssel 7 im Baum in Abb. 4.2. Schreiben Sie ihre Suche wie oben auf.*

4.3.4 Einfügen

Die Möglichkeit ein Element dem Set hinzuzufügen, erfordert das Einfügen eines Schlüssel in einen Suchbaum. Wo muss der Schlüssel eingefügt werden? Genau dort wo er gesucht wird. Damit der Schlüssel dort eingfügt wird, gibt es folgendes Verfahren. Zuerst nach dem Schlüssel suchen. Die Suche kann in einem leeren Baum scheitern, weil der Schlüssel nicht im Baum vorhanden ist. Dann wird der leere Baum mit einem Knoten ersetzt, der den einzufügenden Schlüssel besitzt. Ist die Suche nicht gescheitert, dann ist der Schlüssel schon im Baum vorhanden und es muss nichts gemacht werden. Es ist leicht ersichtlich, dass der Schlüssel richtig eingeordnet wird.

Einfügen Beispiel

Der Algorithmus zum Einfügen des Schlüssels 11 auf dem Baum in Abb. 4.2 ergibt folgende Schritte.
Suchen von Schlüssel 11:
11 ist grösser 10 → Suche geht im rechten Teilbaum der an Knoten 12 Wurzelt weiter.
11 ist kleiner 12 → Suche geht im linken Teilbaum weiter.
Leerer Baum → Suche beendet, Schlüssel ist nicht im Baum.
Einfügen von Schlüssel 11:
Suche endete mit leerem Baum → ersetze leeren Baum durch Knoten mit Schlüssel 11.
Einfügen ist beendet.

Der resultierende Baum ist in Abb. 4.3 dargestellt.
Soll in einen leeren Suchbaum ein Schlüssel eingefügt werden, wird der leere Baum durch einen Knoten mit dem Schlüssel ersetzt. Dieser Knoten ist dann die Wurzel des Baums.

Aufgabe 4.5 *Erstellen Sie Suchbäume aus den folgenden Zahlenfolgen.*

1. *5 8 3 1 9 6 4*
2. *9 6 4 2 3 1*
3. *4 3 5 2 1 6 7*

4.3.5 Entfernen

Das Entfernen eines Schlüssel aus dem Suchbaum ist etwas komplizierter. Zuerst muss der zugehörige Knoten im Baum gesucht werden. Ist der Schlüssel nicht vorhanden, kann nichts entfernt werden und das Entfernen ist beendet. Wird der Knoten gefunden, gibt es vier verschiedene Fälle. In Abb. 4.3 sind die vier Fälle markiert.

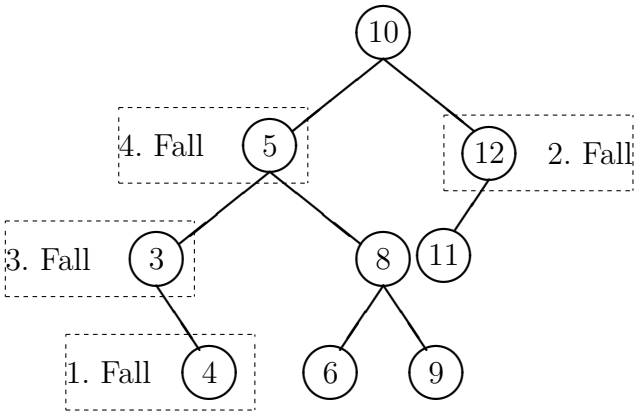


Abbildung 4.3: Der zu entfernende Schlüssel kann in vier verschiedenen Konfigurationen auftreten.

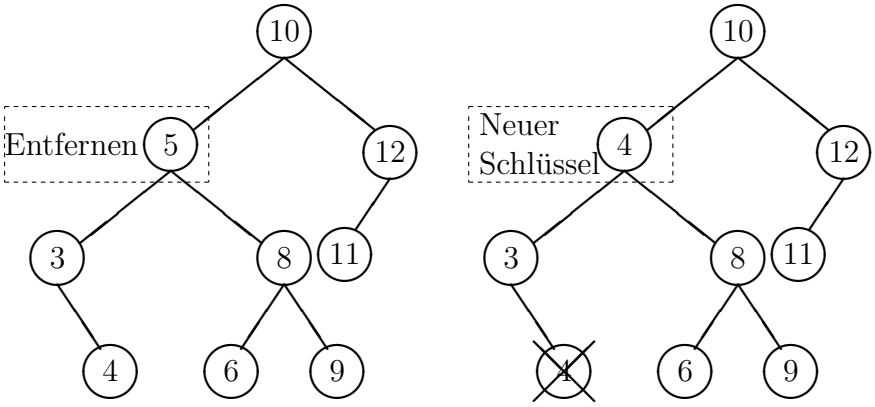


Abbildung 4.4: Entfernung eines Knoten mit zwei Kindern.

Beim ersten Fall ist der zu entfernende Schlüssel in einem Blatt. Blätter können ohne weitere Komplikationen durch den leeren Baum ersetzt werden. Die Ordnung der Knoten im Baum bleibt beim Entfernen eines Blattes immer intakt.
Beim zweiten und dritten Fall ist der zu entfernende Schlüssel in einem Knoten mit genau einem Kind. Knoten mit einem Kind sind auch ziemlich leicht zu entfernen. Der Knoten wird entfernt und sein einziges Kind ersetzt den Knoten bei seinem Vater. Das Kind des Knotens rückt nach oben an seine Stelle. Dieses Kind ist auch wieder die Wurzel eines Teilbaums. In jedem Fall bleibt die Ordnung im Suchbaum erhalten.

Symetrischer Vorgänger. Damit der vierte Fall gelöst werden kann, wird der *symmetrische Vorgänger* eines Knoten benötig. Der symmetrische Vorgänger eines Knoten mit Schlüssel s ist der Knoten mit Schlüssel s', wenn es keinen Schlüssel s'' im Baum gibt, so dass gilt $s' < s'' < s$. Der Knoten mit dem grössten Schlüssel, der kleiner ist wie der Schlüssel in unserem Knoten, heisst symmetrischer Vorgänger. Hat ein Knoten einen nicht leeren linken Teilbaum, dann ist der Knoten mit dem grössten Schlüssel in diesem Teilbaum

sein symmetrischer Vorgänger. Im Suchbaum hat der Knoten, der am weitesten rechts ist, den grössten Schlüssel.

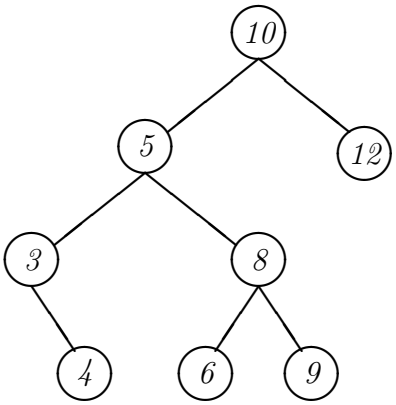
Symetrischer Vorgänger Beispiel Der Suchbaum in Abb. 4.3 hat einen Knoten mit Schlüssel 5. Der symmetrische Vorgänger ist nach obiger Aussage der grösste Knoten in Teilbaum mit Wurzel 3. Der grösste Schlüssel in diesem Teilbaum ist die 4. Somit ist der Knoten mit dem Schlüssel 4 der symmetrische Vorgänger vom Knoten mit dem Schlüssel 5.
Für einen Knoten mit einem linken Kind lässt sich somit einfach der sym. Vorgänger finden. Der Knoten mit dem grössten Schlüssel in einem Suchbaum kann kein rechtes Kind haben. Hätte er ein rechtes Kind, dann wäre der Schlüssel im Kindknoten grösser und der Knoten hätte nicht den grössten Schlüssel. Dies folgt direkt aus der Ordnung im Suchbaum. Somit gilt, dass ein symmetrischer Vorgänger maximal ein linkes Kind haben kann. Dies gilt aber nur, falls der Ausgangs-Knoten ein linkes Kind hat.

Aufgabe 4.6 *Sie haben jetzt neu den Begriff des symetrischen Vorgängers kennen gelernt. Suchen Sie auf die beschriebene Weise den symetrischen Vorgänger des Knotens mit Schlüssel 10 im Baum auf Abb. 4.3.*
Überprüfen Sie, ob der gefundene sym. Vorgänger ein rechtes Kind hat.

Aufgabe 4.7 *Der symetrische Vorgänger des Knoten mit dem Schlüssel 6 wäre der Knoten mit Schlüssel 5 im Baum auf Abb. 4.3. Weshlab kann der Knoten mit Schlüssel 5 ein rechtes Kind haben, obwohl es der sym. Vorgänger von Knoten mit dem Schlüssel 6 ist?*

Beim vierten Fall hat der zu entfernende Knoten zwei Kinder. Die Eigenschaften des symmetrischen Vorgängers eines solchen Knoten lassen diesen vierten Fall auf einen der ersten zwei Fälle reduzieren. Der Schlüssel des symmetrischen Vorgängers wird in den zu entfernenden Knoten kopiert. Danach kann im linken Teilbaum der symmetrische Vorgänger gelöscht werden. Das entfernen des symmetrischen Vorgängers kann nur einer der ersten beiden Fälle sein.
Abb. 4.4 zeigt die Situation des vierten Falles. Der Knoten mit Schlüssel 5 soll gelöscht werden. Der Schlüssel des sym. Vorgängers wird in den Knoten kopiert. Danach kann im linken Teilbaum der sym. Vorgänger gelöscht werden. Dies entspricht hier dem ersten Fall.

Aufgabe 4.8 *Entfernen von Schlüsseln aus einem Suchbaum. Aus dem unten stehenden Suchbaum sollen die Schlüssel 9, 5 und 10 in dieser Reihenfolge entfernt werden. Zeichnen Sie den vollständigen Suchbaum nach dem Entfernen eines jeden Schlüssels auf.*



Sie haben jetzt die drei Wörterbuchoperationen mit Hilfe von Suchbäumen realisiert. Die Wörterbuchfunktionen können auch auf einer geordneten Liste durchgeführt werden.

Aufgabe 4.9 *Entwerfen Sie die Algorithmen für die drei Wörterbuchoperationen auf einer geordneten Liste. Betrachten Sie Anzahl Elemente die Sie bei der Ausführung der Operationen traversieren müssen. Vergleichen Sie die Anzahl der Elemente mit der Anzahl Elementen im Baum.*

4.4 Zusammenfassung

Folgende Themen haben wir in diesem Kapitel behandelt.

- Suchbäume
  Wir verwenden sie für veränderliche Mengen von Schlüsseln.
  Sie besitzen eine Ordnung auf den Schlüsseln.

- Wörterbuchoperationen

  – Suchen
    Mit den Definitionen von Suchbäumen wird die Suche sehr einfach.

  – Einfügen
    Einfügen ist ähnlich wie Suchen, wir fügen einen Schlüssel dort ein, wo wir ihn suchen würden.

  – Entfernen
    Beim Entfernen gibt es vier Fälle, die unterschiedlich behandelt werden müssen. Für einen benötigt man den symmetrischen Vorgänger.

4.5 Lösungen zu den Aufgaben

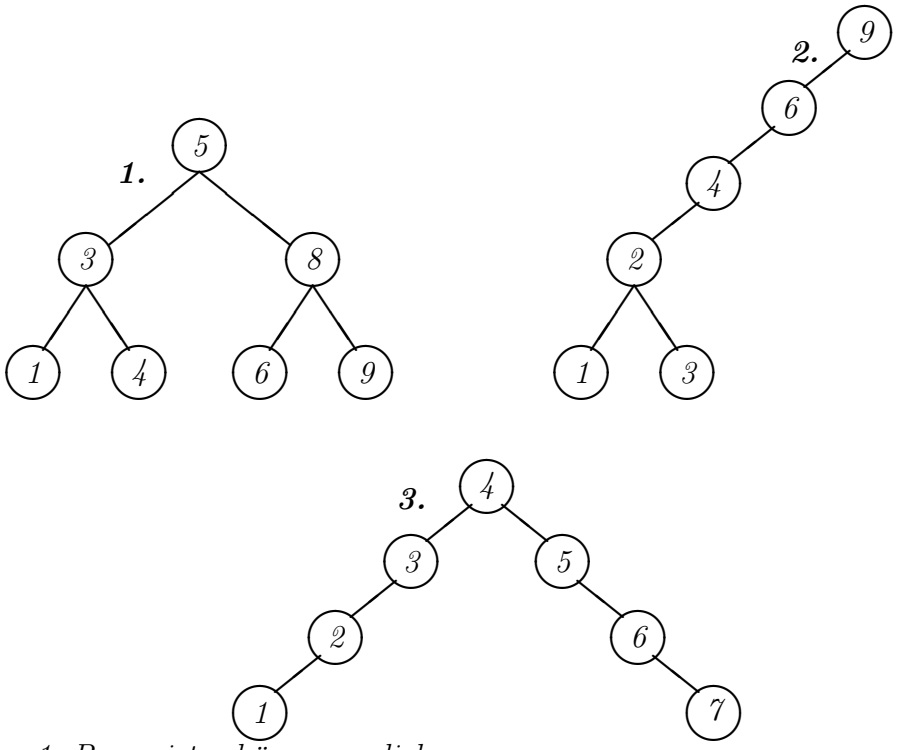
Antwort 4.1 *Die Wörterbuchoperationen decken alle benötigten Operationen ab, damit eine Menge verwaltet werden kann. Sind die Wörterbuchoperationen vorhanden, dann kann damit leicht ein System zur Verwaltung von Daten erzeugt werden. Sie sind ein zentraler Baustein vieler Anwendungen.*

Antwort 4.2 *Zu den drei Bäumen:*
1. Baum: Ist kein korrekter Suchbaum, da der Wurzelknoten 11 drei Kinder hat anstatt nur zwei. Wenn man den Knoten 12 oder den Knoten 15 entfernen würde, dann wäre es ein korrekter Suchbaum, da sonst die Anordnung der Knoten im Baum korrekt ist.
2. Baum: Ist kein korrekter Suchbaum, da der Knoten 6 im linken Teilbaum von 5 ist, und dort nur Knoten mit kleinerem Schlüssel sein dürfen. Wir können diese beiden Knoten vertauschen und erhalten so einen korrekten Suchbaum.
3. Baum: Korrekt.

Antwort 4.3 *Die inorder Traversierung gibt die Schlüssel in aufsteigender Ordnung aus. Dies ist leicht anhand der Definitionen ersichtlich. Die inorder Traversierung traversiert immer den linken Teilbaum, dann den Knoten selbst und danach den rechten Teilbaum. Der Suchbaum ist wie folgt geordnet: Für jeden Knoten sind im linken Teilbaum die kleineren Schlüssel und im rechten Teilbaum die Grösseren. Der Knoten selbst liegt dazwischen. Also wird mit der inorder Traversierung der Suchbaum aufsteigend geordnet ausgegeben.*

Antwort 4.4 *Die Suche verläuft genau gleich wie in den gesehenen Beispielen. Sie endet im rechten Teilbaum des Knoten mit dem Schlüssel 6, da dies ein leerer Baum ist. Der Schlüssel ist nicht im Baum.*

Antwort 4.5 *Die drei Suchbäume*

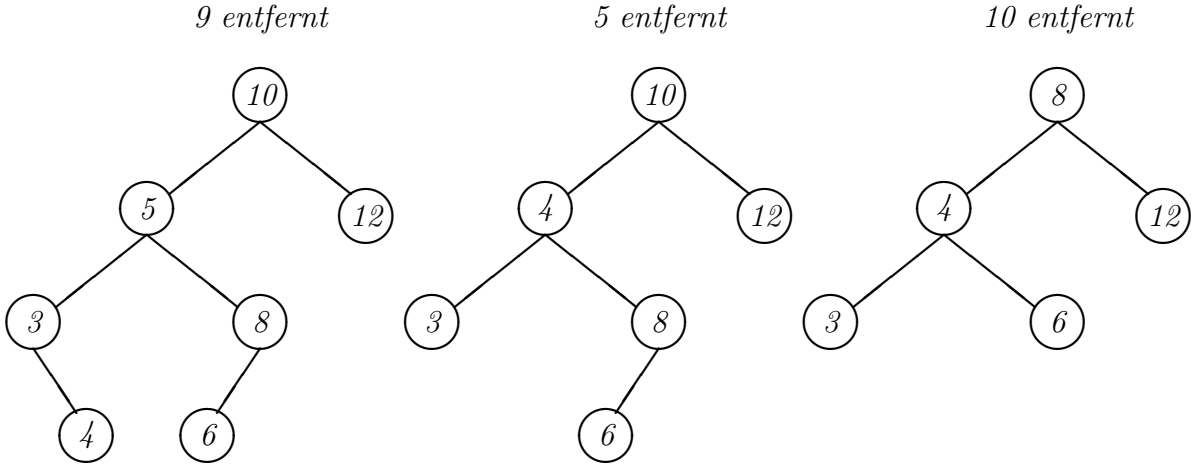


Der 1. Baum ist schön ausgeglichen.
Der 2. Baum ist beinahe zu einer Liste verkommen. Dies zeigt schön die Beziehung zwischen Baum und Liste. Der Baum ist eine Verallgemeinerung der Liste, oder umgekehrt, die Liste ist ein Spezialfall des Baumes.
Der 3. Baum hat auch eine etwas markante Form. Sie haben hier gesehen, dass die Form eines Suchbaumes durch die Reihenfolge des Einfügens der Schlüssel bestimmt wird

Antwort 4.6 *Der sym. Vorgänger vom Knoten mit Schlüssel 10 ist der Knoten mit Schlüssel 9. Wie behauptet, hat diser Knoten kein rechtes Kind.*

Antwort 4.7 *Ein sym. Vorgänger darf nur dann kein rechtes Kind haben, wenn der Ausgangsknoten (6) ein linkes Kind hat. Da aber der Knoten mit Schlüssel 6 kein Linkes Kind hat, kann der zugehörige sym. Vorgänger ein rechtes Kind haben.*

Antwort 4.8 *Die drei entstehenden Bäume sehen wie folgt aus.*



Antwort 4.9 *Die Algorithmen sind sehr einfach auf einer geordneten Liste.*

Suchen *Gehe durch die Liste bis der Schlüssel gefunden ist, oder ein grösserer Schlüssel erreicht ist.*

Einfügen *Suche und an erreichter Position einfügen.*

Entfernen *Suche und falls gefunden, dann entferne Knoten.*

Bei der Liste müssen im Durchschnitt etwa die halben Elemente traversiert werden. Bei dem Baum ist die Anzahl der zu traversierenden Elemente im Durchschnitt unter diesem Wert. Im Idealfall müssen im Schnitt nur Logarithmisch viele Elemente traversiert werden.

Kapitel 5

Implementation Suchbäume

5.1 Übersicht

Sie haben im letzten Kapitel gesehen, wie Suchbäume funktionieren. In diesem Kapitel werden wir das Wissen über die Suchbäume verwenden, um einen zu implementieren. Durch die Reduktion auf grundsätzliche Eigenschaften der Suchbäume werden wir eine Struktur erarbeiten, die es uns ermöglicht, die Operationen sehr einfach zu implementieren.
Die Operationen werden dann von Ihnen in einem Zweierteam implementiert.

5.1.1 Was lernen Sie hier?

Sie lernen hier eine Struktur zu implementieren, die Sie in Ihren Programmen immer wieder verwenden können. Sie sehen auch, wie sich eine gut gewählte Grundstruktur positiv auf die Implementation der Methoden auswirken kann.

5.2 Lernziele

Nachdem Sie dieses Kapitel durchgelesen haben,

verstehen Sie, warum die Implementation für Suchbäume anders ist, als für Etikettenbäume.

kennen Sie zwei neue Programmierkonstrukte, Design Patterns, das Singleton Patter und das Stat Pattern

kennen Sie eine mögliche Implementation für Suchbäume.

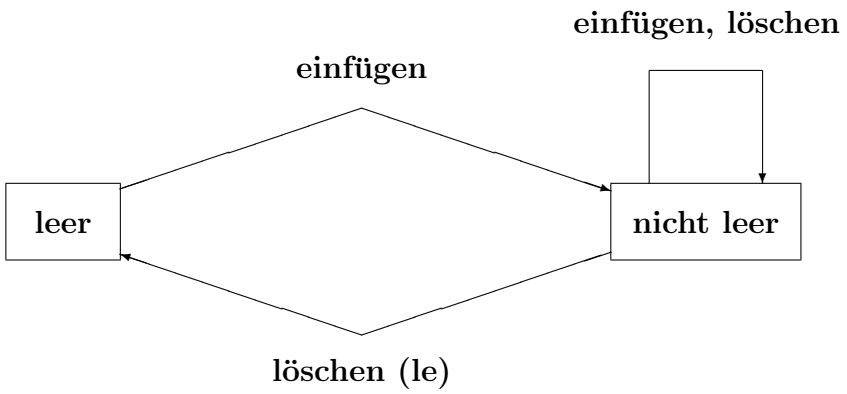


Abbildung 5.1: Die zwei Zustände in der ein Baum sein kann.

5.3 Grundlegende Eigenschaft eines Knoten

Die Suchbäume könnten mit der Technik aus Kapitel 3 implementiert werden. Da der Anwender die interne Anordnung der Knoten nicht benötigt, soll er keinen Zugriff auf die Anordnung haben. Deshalb verwenden wir für den Suchbaum ein anderes Design. Wir wollen hier ein Design finden, welches die Implementation der Methoden nochmals vereinfacht.

Damit wir eine gute Datenstruktur für die Implementation erreichen können, müssen wir die grundlegenden Eigenschaften der Suchbäume finden.

Nach Definition kann ein Binärbaum leer sein oder aus einer Wurzel bestehen die ein linkes und rechtes Kind hat. Die Kinder müssen nach der rekursiven Definition wieder Binärbäume sein. Da es leere und nicht leere Bäume gibt, kann so jeder beliebige Baum der Ordnung zwei erzeugt werden.

Betrachten wir die zwei Zustände eines Baumes etwas näher. Die Abb. 5.1 zeigt die zwei Zustände zusammen mit den Zustandsübergängen. Wie gesagt kann ein Baumm leer sein. Der leere Baum ist im Zustand *leer*. Wird im leeren Baum einen Knoten (Schlüssel) eingefügt, so wechselt der Zustand nach *nicht leer*. Im Zustand *nicht leer* gibt es verschiedene Übergänge. Beim Einfügen eines Knoten macht der Baum im *nicht leer* Zustand keinen Zustandswechsel. Beim Löschen ist die Situation nicht so klar. Wird in einem *nicht leer* Baum der letzte Knoten gelöscht *löschen (le)*, so geht er in den Zustand *leer*. Sind noch weitere Knoten vorhanden, so bleibt der Baum beim Löschen im gleichen Zustand.

Aufgabe 5.1 *Betrachten Sie die Zustände in denen ein Suchbaum sein kann genau. Lassen sich diese Zustände auch auf eine Liste übertragen? Beschreiben Sie die Zustände der Liste.*

Was bringen uns diese Zustände bei der Implementation? Um diese Frage zu beantworten, müssen wir die Operationen betrachten, die auf dem Baum ausgeführt werden sollen. In der Tabelle 5.1 sind die drei Wörterbuchoperationen und das Traversieren aufgeführt. In den beiden rechten Kolonnen ist beschrieben, was in den Zuständen zur

Operation	*leer*	*nicht leer*
Suchen	Suche beendet, nicht gefunden	überprüfen ob Schlüssel gefunden ist, sonst weitersuchen
Einfügen	neuen Knoten erstellen und hier einfügen	wie bei der Suche verfahren
Entfernen	der Knoten ist nicht vorhanden, das Entfernen ist beendet	wie bei Suche, ist der Schlüssel gefunden, entferne ihn
Traversieren	nichts machen	einfach traverseiren

Tabelle 5.1: Was muss in den Zuständen gemacht werden.

Erfüllung der zugehörigen Operation gemacht werden muss. Wie Sie sehen können, sind die Aufgaben in den beiden Zuständen ziemlich verschieden. So muss für jede Operation überprüft werden, in welchem Zustand sich der aktuelle Baum befindet. Ist der Zustand erkannt, müssen die notwendigen Schritte durchgeführt werden. Können bei der Implementation die zwei verschieden Zustände getrennt werden, wir es um einiges übersichtlicher.

Das *State-Pattern* [1], auf Deutsch Zustands-Muster, ist eine Programmiertechnik die auf dieses Problem zugeschnitten ist. Es erlaubt uns auf einfache Weise die Operationen für jeden Zustand einzeln zu implementieren. Das getrennte Implementieren der Methoden reduziert die Anzahl der notwendigen Kontrollstrukturen. Die Entscheidung in in welchem Zustand sich der Knoten befindet wird vom State-Pattern getroffen. Im State-Pattern wird das Objekt, welches mehrere Zustände hat, als *abstrakte Klasse* implementiert. Für jeden Zustand wird eine abgeleitete Klasse erstellt. In den abgeleiteten Klassen werden die Methoden jeweils nur für den zugehörigen Zustand implementiert. Gibt es Gemeinsamkeiten in allen Zuständen, dann können diese bereits in der abstrakten Klasse implementiert werden. Ändert ein Objekt seinen Zustand, so wird die aktuelle Instanz durch eine passende Instanz der Klasse des neuen Zustandes ersetzt. Dies ist möglich, da alle konkreten Klassen von der gleichen abstrakten Klasse erben. Beim Ausführen des Programms wird somit automatisch die Implementation des aktuellen Zustandes verwendet.

Aufgabe 5.2 *Damit Sie das State-Pattern implementieren können müssen Sie das Prinzip von abstrakten Klassen und Polymorphismus verstehen. Diskutieren Sie mit ihrem Partner diese zwei Themen. Nehemen Sie gegebenenfalls ihre Unterlagen zu diesen Themen zur Hilfe.*

5.4 Grundstruktur und Interface

Das State-Pattern vereinfacht die Implementation der Methoden. Wir wollen nun eine Struktur definieren, die diese Einfachheit weiter zieht. Der binäre Suchbaum soll mit der Klasse `BinSuchbaum` implementiert werden. Da ein Suchbaum entweder leer sein kann oder nicht, beinhaltet die `BinSuchbaum` Klasse nur einen abstrakten Knoten. Siehe Abb. 5.2. Der Code für die Felder.

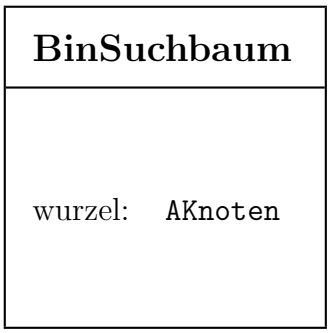


Abbildung 5.2: BinSuchbaum Klasse mit dem AKnoten.

```
/** Ein abgeschlossener (encapsulation) Suchbaum
 * implementiert mit dem State- und Sigleton-Pattern.
 */
class BinSuchbaum
{
  /** Die Wurzel (Knoten)
   * Ein Baum ist durch die Wurzel und seine Kinder definiert
   * Für den leeren Baum ist die Wurzel durch den LeerenKnoten
   * repräsenteirt.
   */
  private AKnoten _wurzel;
}
```

Besteht der Baum aus mehr wie nur einem Knoten, wird genau nach der rekursiven Definition verfahren. Es werden links und rechts wieder Suchbäume angehängt. Der abstrakte Knoten ist in der Klasse `AKnoten` implementiert. Mit Hilfe der Klasse `AKnoten` wird das State-Pattern implementiert. Es werden zwei abgeleitet Klassen erstellt, für jeden Zustand eine. Die Klasse `LeererKnoten` für den leeren Baum und die Klasse `NichtLeererKnoten` für den nicht leeren Baum. Die Abb. 5.3 zeigt die Knotenklassen und ihre Beziehungen. Verweist ein Suchbaum Objekt auf einen Knoten vom Typ `LeererKnoten` so ist der Baum leer. Ein leerer Baum hat keine Unterbäume, deshalb braucht die Klasse `LeererKnoten` keine Verweise auf weitere Suchbäume. Ist der Suchbaum nicht leer, so verweist das Baumobjekt auf einen nicht leeren Knoten. Die Klasse `NichtLeererKnoten` hat drei Felder. Den Schlüssel und das linke und rechte Kind vom Typ `BinSuchbaum`. Die Abb. 5.4 zeigt die Struktur eines Suchbaums mit zwei Knoten.

Der Code für die Struktur der Knoten Klassen.

```
  /** Abstrakter Knoten
   * Suchbaum referenziert diesen Knoten als Wurzel.
   * Die abstrakte Klasse wird für das State-Pattern
   * verwendet.
   * State-Pattern ermöglicht eine einfache
```

```
 *  funktionsbezogene Implementierung.
 */
static private abstract class AKnoten
{
  // keine Felder
}
```

Der abstrakte Knoten hat keine Felder.

```
/** Leerer Knoten
 * Diese Klasse repräsentiert den leeren Knoten.
 * Singleton Klasse -> es gibt nur eine Instanz,
 *   alle leeren Bäume sind gleich.
 * Implementiert alle Operationen auf dem leeren Baum.
 */
static private class LeererKnoten extends AKnoten
{
  // statisches Feld für Singleton Pattern
  private static AKnoten _einmalig = null;

  /** Zugriff auf einzige Instanz */
  public static AKnoten singleton() {
    if (_einmalig == null) {
      _einmalig = new LeererKnoten();
    }
    return _einmalig;
  }
}
```

Der leere Knoten hat nur ein statisches Feld für das Singleton-Pattern.

```
/** NichtLeererKnoten
 * Zustand nicht leerer Knoten. Beinhaltet den
 * Schlüssel.
 * Implementiert alle Operationen auf einem Knoten
 */
static private class NichtLeererKnoten extends AKnoten
{
  // Felder
  /** Das linke Kind */
  private BinSuchbaum _links;
  /** Das rechte Kind */
  private BinSuchbaum _rechts;
```

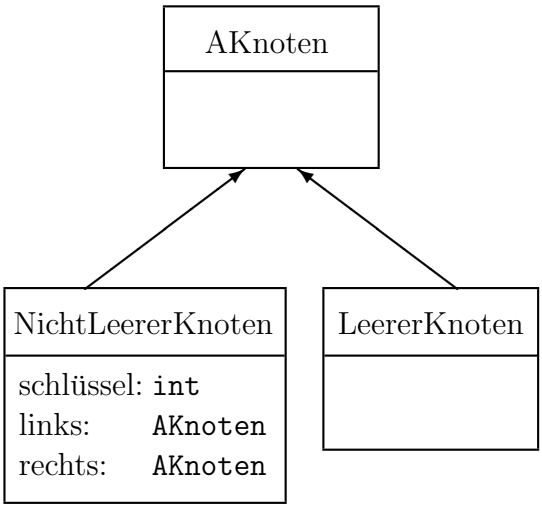


Abbildung 5.3: Die abstrakte Klasse `AKnoten` mit den 2 Implementationen. Für jeden Zustand eine Klasse.

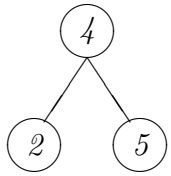
```
    /** Der Schlüssel */
    private int _schluessel;
  }
```

Der nicht leere Knoten verweist auf den linken und rechten Teilbaum. Er hat auch einen Schlüssel.

Die Instanzen der Klasse `LeererBaum` haben keine Felder. Sie repräsentieren auch *nichts*. Da sie keine Felder haben, sind alle Instanzen ununterscheidbar. Wie in Abb. 5.4 ersichtlich ist, gibt es im Beispielbaum nur eine Instanz der Klasse `LeererKnoten`. Alle leeren Suchbäume verweisen somit auf die gleiche Instanz. Dies wirkt sich positiv auf die Effizenz des Codes aus. Die Einzigartikeit der `LeererKnoten` Klasse wird erreicht, indem sie als eine *Singleton* Klasse implementiert wird. Singleton steht für Einzigartigkeit. Eine singleton Klasse hat ein statisches Feld das auf die einzige Instanz dieser Klasse verweist. Durch Aufruf der statischen Methode `singleton()` erhält man Zugriff auf diese Instanz.

Die Knoten Klassen werden als private eingenistete Klassen von `BinSuchbaum` implementiert. Daraus folgt, dass diese Klassen ausserhalb von `BinSuchbaum` nicht definiert sind. Dies stellt die Abgeschlossenheit der Baumes sicher.

Aufgabe 5.3 *Zeichenen Sie Die Objektstruktur des folgeneden Suchbaumes. Nehmen Sie die Abb. 5.4 als Vorlage. Fügen Sie zu den NichtLeererKnoten Objekte noch den zugehörenden Schlüssel ein.*



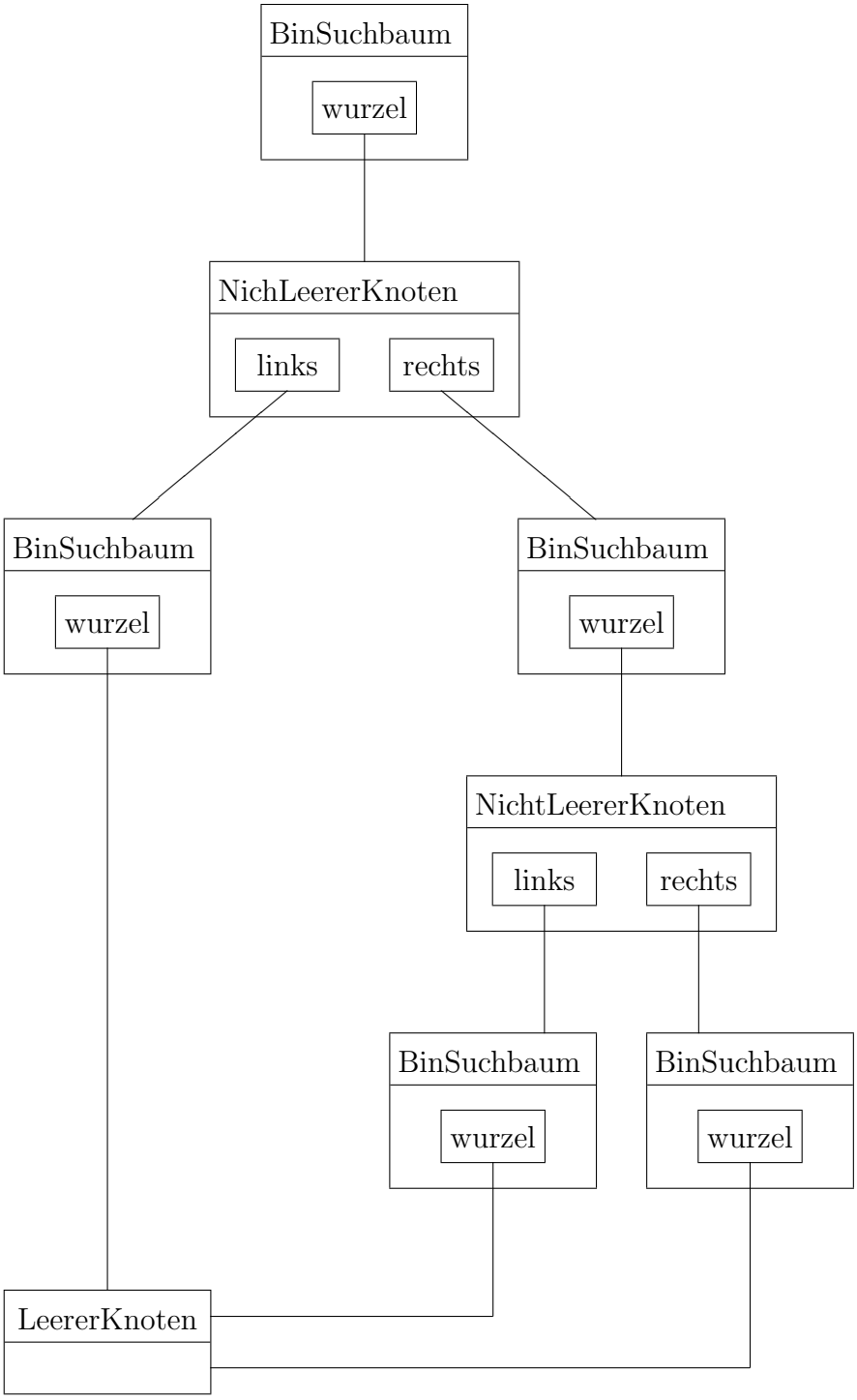


Abbildung 5.4: Ein Beispiel Suchbaum

5.4.1 Interface

Zum Interface gehören alle Operationen, die wir auf dem Suchbaum erlauben wollen. Der Konstruktor, die drei Wörterbuchoperationen und die Inordertraversierung. Die Klasse `AKnoten` ist von aussen nicht Sichtbar und gehört deshalb nicht zum Interface des Bin-Suchbaumes.

Das Interface der Klasse `AKnoten` wird nur von der Baum-Klasse verwendet. Dieses bestimmen wir bei der Implementation.

Aufgabe 5.4 *Schreiben Sie das Interface der Klasse BinSuchbaum auf. Versuchen Sie es möglichst einfach zu halten.*

5.5 Implementation der Methoden

Die Struktur mit den zwei Klassen gibt etwas mehr Schreibarbeit, erlaubt es uns aber jetzt nur noch auf die Implementation der Funktionalität zu achten.

Die Alogrithmen aus Kapitel 4 aller Operationen werden in zwei Teile geteilt. Einen Teil für den nicht leeren Knoten und einen für den leeren Knoten. Die Aufteilung der Algorithmen ist in Tabelle 5.1 beschriebn.

Betrachten wir die Methode `suchen(int k)`.

Code der Klasse `LeererKnoten`. Die Suche ist in einem leeren Baum angelangt. Kein Schlüssel ist in einem leeren Baum vorhanden. Die Suche ist fehlgeschlagen. Dafür wird folgender Code benötig.

```
public boolean suchen(int k) {
  return false;
}
```

Code der Klasse `NichtLeererKnoten`. Die Suche ist in einem Knoten. Falls der Schlüssel mit dem gesuchten übereinstimmt ist die Suche erfolgreich beendet. Entsprechen sich die Schlüssel nicht, so muss im linken oder rechten Teilbaum weiter gesucht werden. Der folgende Code macht genau dies.

```
public boolean suchen(int k) {
  if (k == _schluessel) { return true; }
  if (k < _schluessel) { return _links.suchen(k); }
  else { return _rechts.suchen(k); }
}
```

Ist ihnen aufgefallen, dass der Code nichts anderes macht, wie die Schlüssel zu vergleichen und dann die erforderlichen Aktionen ausführt? Es muss nirgends überprüft werden, ob es sich jetzt um ein Blatt handelt oder nicht. Es werden auch keiner Referenzen auf `null` geprüft.

Aufgabe 5.5 *Bei den Dateien im Ordner* Aufgaben *können Sie die Datei* binsuchbaum.java *finden. Die Struktur und die Methodenrümpfe sind alle vorgegeben. Vervollständigen Sie mit ihrem Partner die Implementation der Methoden* `einfügen(int k)`, `istLeer()` *und* `printInOrder()`. *In der Klasse* `BinSuchbaum` *gibt es nichts mehr zu tun. Sie müssen nur die zugehörigen Teile in der Knoten Klassen vervollständigen.*

Damit Sie die *entfernen(int k)* Methode implementieren können, schauen Sie die in Kapitel 4 vorgestellte Hilfsfunktion an. Die Methode heisst `entferneGroesster(BinSuchbaum b)` und ist betreits implementiert. Diese Methode entfernt den Knoten mit dem grössten Schlüssel und gibt den Schlüssel zurück. Diese Methode darf nicht auf dem leeren Baum aufgerufen werden.

Aufgabe 5.6 *Vervollständigen Sie mit ihrem Partner die Methode* `entfernen(int k)`. *Schauen Sie sich die Hilfsmethode nochmals an. Und vergewissern Sie sich, dass Sie den Algorithmus noch kennen. Sind Sie unsicher schauen Sie nochmal in Kapitel 4 nach. Testen Sie ihre Implementation mit dem Testprogramm* Test2.java.

Sie haben jetzt eine funktionierende Implementation eines binären Suchbaumes. Diesen Suchbaum können Sie nun in in Ihren Anwendungen verwenden. Da die Knoten Klassen vor dem direkten Zugriff geschützt sind, verändert sich das Verhalten in einer anderen Umgebung nicht.

5.6 Schlussbemerkung

Dieser Suchbaum erfüllt alle Anforderungen die wir in Kapitel 4 an die Implementation stellten. Durch die Abgeschlossenheit kann er ohne Gefahr in vielen Anwendungen eingesetzt werden. Die meisten Computerprogramme verwalten Daten. All diese Programme könnten einen Baum dazu verwenden, wenn sie einen Schlüssel auf den Daten definieren können. Obwohl es so viele Anwendungen für die Suchbäume gibt, werden Sie mehr strukturerhaltende Etikettenbäume implementieren. Und das nur weil diese Bäume sellten wiederverwendet werden.

Als Weiterführung könnten Sie sich mit der Effizienz von Suchbäumen befassen. Dieses Thema ist wichtig, da der Geschwindikeitsgewinn gegenüber andere Strukturen den Suchbaum so wichtig macht.

5.7 Lösungen zu den Aufgaben

Antwort 5.1 *Bei einer Liste gibt es die genau gleichen Zustände. Das Diagramm in Abb. 5.1 könnte auch von einer Liste stammen.*

Antwort 5.2 *Diese Aufgabe müssen Sie selbst bearbeiten. Eine Lösung würde den Rahmen dieses Leitprogrammes sprengen.*

Antwort 5.3 *Die Objektstruktur ist identisch mit der aus Abb. 5.4 ausser, dass links auch noch ein NichtLeererKnoten am obersten NichtLeerenKnoten ist.*

Antwort 5.4 *Die Folgenden Operationen sind auf dem Suchbaum erlaubt.*

```
class BinSuchbaum
{
  /** Die Wurzel Knoten
   * Ein Baum ist durch die Wurzel und seine Kinder definiert
   * Für den leeren Baum ist die Wurzel durch den LeerenKnoten
   * repräsenteirt.
   */
  private AKnoten _wurzel;

  /** Erzeugt leerer Baum */
  public BinSuchbaum() { ... }

  /** Ist der Schlüssel im Baum (Suche)
    * @param k Schlüssel
    */
  public boolean suchen(int k) { ... }

  /** Fügt Schlüssel ein
   * @param k Schlüssel
   */
  public void einfuegen(int k) { ... }

  /** Entfernt Schlüsel
   * @param k Schlüssel
   */
  public void entferne(int k) { ... }

  /** Ist Baum leer
   * @return true false leer
   */
  public boolean istLeer() { ... }
```

Antwort 5.5 *Eine Lösung ist in der Datei* binsuchbaum.java *im Ordner* Loesungen. *Vergleichen Sie ihre Lösung mit der aus der Datei. Diskutieren Sie Unklarheiten mit einer anderen Gruppe.*

Antwort 5.6 *Eine Beispiellösung ist in der Datei* binsuchbaum.java *im Ordner* Loesungen. *Vergleichen Sie ihre Lösung mit der aus der Datei. Diskutieren Sie Unklarheiten mit einer anderen Gruppe.*

Anhang A

Kapitel-Test für den Tutor

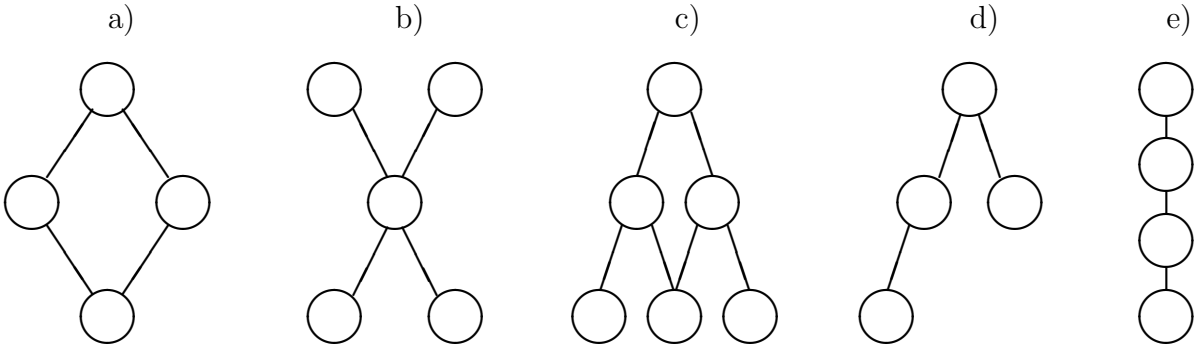
Vier Aufgaben für jedes Kapitel. Die Aufgaben sind jeweils auf einer eigenen Seite, damit sie den Schülern auch schriftlich abgegeben werden können. Die Aufgaben dienen als Kapitel Test. Können die Schülerinnen die Fragen beantworten, dann sind sie bereit für das nächste Kapitel.

Die zugehörenden Antworten befinden sich immer eine Seite hinter den Fragen. Die Antworten sind nur mögliche Antworten. Es gibt häufig verschieden Arten eine Frage zu beantworten.

A.1 Kapitel 1

Aufgaben

Aufgabe 1 Welche der folgenden Graphen stellen Bäume dar? Begründen Sie ihre Aussagen.



Aufgabe 2 Erklären Sie nochmals die 6 folgenden Begriffe mit Ihren eigenen Worten.

- Knoten
- Wurzel
- Blatt
- Vater
- Kind
- Ordnung

Aufgabe 3 Welche Ordnung hat eine Liste?

Aufgabe 4 Weshalb ist der Baum eine gute Datenstruktur? Versuchen Sie diese Frage mit Ihrem Verständnis von Bäumen zu beantworten.

Antworten zu Kapitel 1

Dies sind Antworten die richtig sind oder als richtig zugelassen sein sollen.

Antwort 1 (K 3)

a) Dieser Graph ist kein Baum. Ein Baum kann nach Definition keine Zyklen enthalten. Es ist auf diesem Graph auch nicht möglich, die Vater-Beziehung zu zeigen.

b) Dieser Graph ist ein Baum. Die Struktur lässt sich aus der Definition erstellen. Die Wurzel ist nicht festgelegt. Es kann also ein beliebiger Knoten als Wurzel gewählt werden.

c) Dies ist auch kein Baum. Der Graph beinhaltet Zyklen und lässt sich somit nicht aus der Definition für Bäume erstellen.

d) Dieser Graph ist ein Baum.

e) Dies ist eine Liste. Listen sind auch Bäume. Der Graph ist somit auch ein Baum.

Antwort 2 (K 2) Mögliche Antworten sind:

Knoten Ein Knoten ist ein Element in einem Baum.

Wurzel Eine Wurzel ist ein ausgewählter Knoten eines Baumes. Die Wurzel wird meistens zuoberst im Baum dargestellt. Sie hat keinen Vater.

Blatt Ein Blatt ist ein unterster Knoten eines Baumes. Es hat keine Kinder mehr.

Vater Ein Vater ist der Vorfahre eines bestimmten Knotens.

Kind Ein Kind ist der Nachkomme eines bestimmten Knotens.

Ordnung Die Ordnung eines Baumes ist die maximale Anzahl Kinder die ein Knoten im Baum haben kann.

Antwort 3 (K 3)
Die Liste ist ein Baum, wo jeder Knoten nur ein Kind hat. Somit ist die Liste ein Baum mit der Ordnung 1.

Antwort 4 (K 4)
Dies ist eine sehr subjektive Frage. Der Baum ist eine sehr intuitive Datenstruktur. Er ist aus dem Alltag bekannt. Mit dem Baum ist es möglich viele Informationen in ihrer natürlichen Form abzubilden. Die Daten müssen nicht kompliziert verarbeitet werden.

A.2 Kapitel 2

Aufgaben

Aufgabe 1 In diesem Kapitel haben Sie drei Arten einen Baum zu traversieren kennen gelernt. Die drei Traversierungen wurden auf einem binären Baum eingeführt.
Kann man diese Traversierungen auch auf Bäumen höherer Ordnung sinnvoll definieren? Betrachten Sie jede Traversierung für sich selbst.

Aufgabe 2 Weshalb wird bei den Anforderungen für die Etikettenbäume immer verlangt, dass sich die Struktur des Baumes nicht verändern darf?
Tipp: Überlegen Sie sich was die Information ist, die in solchen Bäumen gespeichert wird.

Aufgabe 3 Weshalb ist bei der Infix-Notation die Klammerung notwendig, wenn sie durch die Inorder-Traversierung erzeugt wurde. Können Sie dies mit Hilfe von Bäumen veranschaulichen?
Tipp: Verschiedene Bäume auf die gleiche ungeklammerte Infix-Notation abbilden.

Aufgabe 4 Können Sie erklären, weshalb sich Bäume gut eignen, um mathematische Ausdrücke darzustellen?

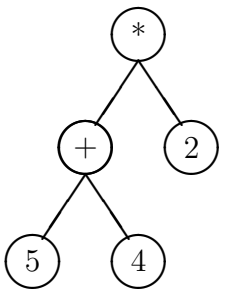
Antworten zu Kapitel 2

Dies sind Antworten die richtig sind oder als richtig zugelassen sein sollen.

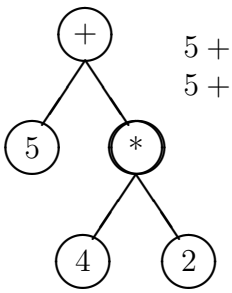
Antwort 1 (K 4-5)
Die Preorder- und Postorder-Traversierungen lassen sich einfach auf Bäume höherer Ordnung definieren. Bei der Preorder-Traversierung wird zuerst die Wurzel verarbeitet und dann werden der Reihe nach alle Unterbäume traversiert.
Bei der Postorder-Traversierung werden zuerst alle Unterbäume der Reihe nach traversiert und danach wird die Wurzel verarbeitet.
Die Inorder Traversierung lässt sich auf Bäumen höherer Ordnung nicht sinnvoll definieren. Es lässt sich nicht schlau sagen, an welcher Stelle die Wurzel abgearbeitet werden soll.

Antwort 2 (K 3)
Die Struktur der Etikettenbäume ist Teil der gespeicherten Information. Die Anordnung der Knoten im Baum steht für Beziehungen zwischen den Daten auf den Etiketten. Angenommen ein Baum repräsentiert einen Stammbaum. Würde bei diesem Baum die Anordnung der Knoten durcheinander geraten, dann wären die gespeicherten Verwandschaftsbeziehungen nicht mehr korrekt. Was natürlich nicht geschehen darf.

Antwort 3 (K 4)
Die verschiedenen Operatoren in der Infix-Notation haben verschiedene Bindungsstärken. Entsteht ein Infix-Ausdruck durch die Inorder-Traversierung, dann wird die Bindungstärke nicht berücksichtigt. Dazu kommt das bei der Inorder-Traversierung strukturelle Informationen vom Baum verloren gehen. Zwei verschiedene Bäume können auf die gleiche ungeklammerte Infix-Notation abgebildet werden. Kleines Beispiel folgt.



$5 + 4 * 2$
$(5 + 4) * 2$



$5 + 4 * 2$
$5 + (4 * 2)$

Antwort 4 (K 4)
Die Bäume sind sehr geeignet mathematische Ausdrücke Abzubilden, da sie die Struktur der Ausdrücke sehr gut wiedergeben.

A.3 Kapitel 3

Aufgaben

Aufgabe 1 Sie haben in diesem Kapitel einen Etikettenbaum implementiert. Können Sie erklären weshalb Etikettenbäume für fast jede Anwendung neu implementiert werden? Warum gibt es keine Implementierung, die man immer wieder verwendet, wie es bei einer Liste der Fall ist?

Aufgabe 2 Können Sie erklären wie die Implementation eines Etikettenbaumes aussieht? Welche Klassen werden benötige? Was für Felder kommen in den Klassen vor?
Wie wird ein solcher Baum aufgebaut, wenn die Implementation gegeben ist?

Aufgabe 3 Sie haben einen Baum für die Repräsentation von mathematischen Ausdrücken erarbeitet. Dieser Baum hatte zwei Kinder das linke und rechte. Können Sie eine kleine Methode einer Knoten Klasse schreiben die eine Inorder-Traversierung macht?
Tipp: Überlegen sie sich gut, wo der Baum zu ende ist.

Aufgabe 4 Sie sollen nun für einen Stammbaum eine Implementation schreiben. Gegeben einen Knoten der eine bestimmte Person beschreibt, sollen folgende Fragen beantwortet werden können.
Wie viele Kinder hat die Person.
Hat die Person eine Tante(Onkel).
Welches sind ihre Nachfahren.
Sie müssen keine Methoden schreiben, welche die obigen Fragen beantworten. Ihre Implementation soll nur eine Datenstruktur zur Verfügung stellen, so dass die Fragen einfach beantwortet werden können.

Antworten zu Kapitel 3

Dies sind Antworten die richtig sind oder als richtig zugelassen sein sollen.

Antwort 1 (K 4)
Die Implementation eines Etikettenbaumes ist immer sehr auf die Anwendung zugeschnitten. Die Bäume bestehen meist nur aus einer Knoten Klasse, die sehr eng mit dem Programm verwoben sind. Dazu sind die Knoten Klassen auch sehr einfach. Es ist nicht möglich eine abgeschlossene Implementation zu erstellen, da die Struktur von aussen zugreifbar sein muss.

Antwort 2 (K 3)
Der Etikettenbaum besteht meist aus nur einer Klasse. Diese Klasse hat ein Feld für die Etikette das Abhängig von den zu speichernden Daten gewählt wird. Dazu kommen die Felder für die Kinder. Eventuell auch eine Liste von Kindern, wenn die Anzahl nicht konstant ist. Ein Feld für einen Verweis zum Vater kann auch noch eingefügt werden.
Der Baum wird von der umgebenen Anwendung aufgebaut, indem die einzelnen Knoten verlinkt werden. Bei Knoten die einen Verweis auf den Vater haben, ist ein eine geeignete Methode zur Verlinkung der Knoten von Vorteil.

Antwort 3 (K 2)
Der Baum endet immer dort, wo ein Kinder-Feld den Wert `null` hat.

```
  public void inorder() {
    if (_links != null) { _links.inorder();}
    System.out.println(_etikette);
    if (_rechts != null) { _rechts.inorder();}
  }
```

Antwort 4 (K 3)
Für die Anzahl Kinder Frage muss es ein Feld oder Liste für die Kinder geben. Für die Frage nach der Tante braucht es ein Feld für den Vater. Daraus folgt folgende Struktur.

```
class StammbaumKnoten {
  StammbaumKnoten _vater;
  LinkedList<StammbaumKnoten> kinder;
  Person person;
}
```

A.4 Kapitel 4

Aufgaben

Aufgabe 1 Sie haben in diesem Kapitel gelernt, dass die Knoten in den Suchbäumen geordnet sind. Können Sie eine Traversierung beschreiben, so dass die Knoten in absteigender Ordnung traversiert werden? Hier geht es um die absteigende Ordnung, nicht die Aufsteigende.

Aufgabe 2 Der Algorithmus zum Entfernen eins Schlüssels aus einem Suchbaum haben Sie in diesem Kapitel angewendet. Können Sie diesen Algorithmus erläutern? Aus welchen Phasen besteht er und welche Fälle gibt es zu beachten?

Aufgabe 3 Könnte man beim Algorithmus für das Entfernen von Knoten an der Stelle des symmetrischen Vorgängers auch einen anderen Knoten verwenden? Und wie würde dann der Algorithmus aussehen.
Tipp: Gibt es einen Knoten der symmetrisch zum symmetrischen Vorgänger ist.

Aufgabe 4 Weshalb werden Suchbäume häufig für die Verwaltung von Schlüssel mengen verwendet. Was ist der Vorteil gegenüber einer Liste.

Antworten zu Kapitel 4

Dies sind Antworten die richtig sind oder als richtig zugelassen sein sollen.

Antwort 1 (K 4)
Die absteigende Ordnung erhält man, mit einer umgedrehten Inorder-Traversierung. Dabei wird zuerst der rechte Teilbaum, dann der Knoten und am Schluss der linke Teilbaum Traversiert.

Antwort 2 (K 2)
1. Suche den Schlüssel.
2. Zwei Fälle.
   a) Schlüssel nicht gefunden dann ist das Entfernen beendet.
   b) Schlüssel gefunden weiter zu 3.
3. Vier Fälle.
   a) Knoten hat kein Kind → Löschen und fertig.
   b) Knoten hat ein linkes Kind → Ersetze Knoten mit Kind fertig.
   c) Knoten hat ein rechtes Kind → Ersetze Knoten mit Kind fertig.
   d) Knoten hat zwei Kinder → mache bei 4 Weiter.
4. Suche symmetrischen Vorgänger.
5. Kopiere den Schlüssel des symmetrischen Vorgänger in den Knoten
6. Entferne symmetrischen Vorgänger.

Antwort 3 (K 5)
Der symmetrische Nachfolger kann die Rolle des symmetrischen Vorgängers übernehmen. Gleicher Algorithmus wie bei Antwort 2. Vorgänger durch Nachfolger ersetzen. Der sym. Nachfolger ist der Knoten mit dem kleinsten Schlüssel im rechten Teilbaum. Es gelten die gleichen Eigenschaften wie beim Vorgänger.

Antwort 4 (K 4)
Die Bäume sind viel effizienter. Bei einer Liste müssen beim Suchen, Einfügen und Entfernen viel mehr Elemente Traversiert werden. Dies ist so, weil sich der Suchbaum aufsplittet.

A.5 Kapitel 5

Aufgaben

Aufgabe 1 Was versteht man unter der Abgeschlossenheit bei der Objekt orientierten Programmierung? Was ist an unserer Implementation des Suchbaumes abgeschlossen?

Aufgabe 2 Weshalb kann die Implementation eines Suchbaumes in vielen Anwendungen verwendet werden?

Aufgabe 3 Bei der Implementation dieses Suchbaumes wurden Design Patterns verwendet. Welchen Vorteil bringt das Zustands-Muster für die Implementierung?

Aufgabe 4 Wie wird bei unserer Implementation das Baumende erkannt?
Bei den Etikettenbäumen wird dies durch die Überprüfung der Kinder-Felder auf `null` erreicht.

Antworten zu Kapitel 5

Dies sind Antworten die richtig sind oder als richtig zugelassen sein sollen.

Antwort 1 (K 2)
Unter Abgeschlossenheit bei der oo Programmierung versteht man, dass nicht auf die interne Struktur einer Klasse oder eines komplexen Gebildes zugegriffen werden kann. Es kann nur über das öffentlich deklarierte Interface auf die Daten zugegriffen werden.
(K 4)
In unserer Implementation sind die ganzen Knoten Klassen vor dem Zugriff geschützt. Die Anwendung kann die Struktur des Suchbaumes nicht direkt manipulieren.

Antwort 2 (K 4)
Der Suchbaum implementiert einen abstrakten Datentyp. Dieser Datentyp ist genau definiert. Der Suchbaum lässt keine Aktionen zu die im ADT nicht definiert sind. Daraus folgt, dass überall wo diese Eigenschaften gefragt sind, der Suchbaum ohne Sorgen eingesetzt werden kann.

Antwort 3 (K 2)
Die Implementation für die Methoden ist um einiges einfacher. Mit dem Zustands-Muster kann die Implementation der Methoden auf die verschiedene Zustände aufgeteilt werden. Für jeden Zustand muss nur die notwendige Funktionalität implementiert werden.

Antwort 4 (K 2)
Bei unserer Implementation gibt es keine Felder die auf `null` überprüft werden müssen. Jeder Knoten verweist wieder auf einen Baum. Ist der Baum zu Ende, dann verweist der Knoten auf einen leeren Baum. Der leere Baum behandelt diese Situation.

Literaturverzeichnis

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides *Design Patterns - Elements of Reusable Object-Oriented Software* USA, 1995 (Addison-Wesley)

[2] T. Ottmann, P. Widmayer *Algorithmen und Datenstrukturen* 2. Auflage, Deutschland, 1993 (Wissenschaftsverlag)

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest *Introduction to Algorithms* USA, 1990 (MIT Press)

[4] A. V. Aho, J. E. Hopcroft, J. D. Ullman *Data Structures and Algorithms* USA, 1983 (Addison-Wesley)

[5] D. Z. Nguyen *Design Patterns for Data Structures* ACM SIGCSE, USA, 1998, pages: 336 - 340

[6] H. Kopka *LaTeXBand 1: Einführung* 3. Auflage, Deutschland, 2000 (Addison-Wesley)

[7] http://java.sun.com/j2se/1.5.0/docs/api/index.html