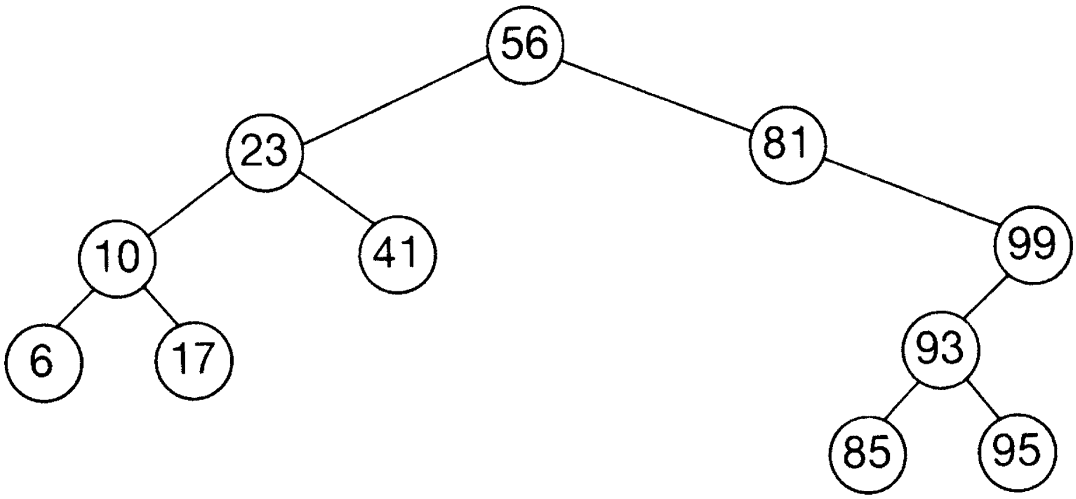
Binäre Suchbäume

Auszug aus:
laubach / knoch:
grundkurs informatik 2
bsv



9.3. Binäre Suchbäume

9.3.1. Einführende Beispiele

Binäre Bäume eignen sich gut zum gezielten Speichern und schnellen Wiederauffinden von Informationen. In früheren Kapiteln wurden Verfahren besprochen, mit denen Daten in linearen Anordnungen gespeichert und gesucht wurden (Felder, Listen). Als schnelles Suchverfahren haben wir bei Feldern das binäre Suchen

und das Hash-Verfahren kennen gelernt. Dabei haben wir gesehen, daß zum schnellen Wiederfinden von Daten eine wichtige Voraussetzung gehört: die Daten müssen vorher in eine passende Anordnung gebracht werden. Binäres Suchen ist nur möglich, wenn die Datenelemente nach einem Schlüssel aufsteigend (oder absteigend) sortiert sind. Eine Suche mit Hilfe eines Hash-Verfahrens erfordert, daß zuvor mit derselben Hash-Funktion gespeichert wurde, mit der nun gesucht wird. In einem binären Baum kann die Suche nach einem Knoteninhalt dann schnell erfolgen, wenn der Baum passend aufgebaut ist und man bei jedem Knoten feststellen kann, in welchem Teilbaum man weitersuchen muß, falls das Suchelement im gerade betrachteten Knoten nicht enthalten ist. Anderenfalls müßte man nämlich den ganzen Baum durchsuchen.

Beispiel

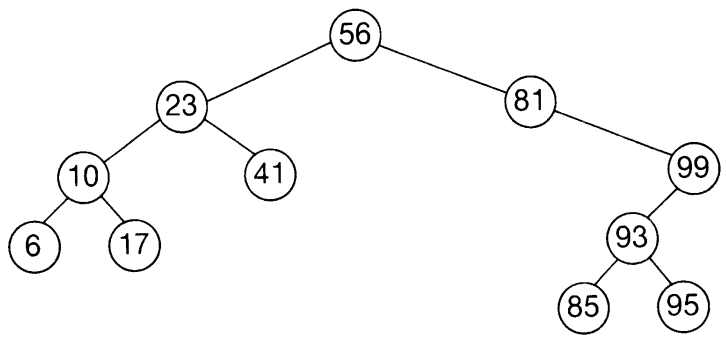
Die einzelnen Fragen in den Knoten des Binärbaumes beim Beruferaten besitzen keine Ordnung in dem gerade beschriebenen Sinn. Wenn man etwa feststellen wollte, ob der Beruf „Maurer" bereits bekannt ist, müßte man den Baum systematisch (z. B. in Preorder) durchlaufen und in den Blättern nachprüfen, ob dort dieser Beruf vorkommt (als letztes Wort der Frage). Der Wissensbaum ist also kein Suchbaum zum schnellen Auffinden einer Information.

Wie bei allen früher besprochenen Suchverfahren nehmen wir an, daß die Datenelemente einen Schlüssel als Komponente enthalten. Im gesamten gespeicherten Datenbestand soll es keine zwei Elemente mit demselben Schlüssel geben. Für den Typ dieses Schlüssels soll eine Ordnung definiert sein, die Vergleiche auf „*kleiner*" bzw. „*größer*" zwischen zwei Werten ermöglicht. Für erläuternde Beispiele werden der Einfachheit halber Datenelemente verwendet, die nur den Schlüssel enthalten. Vorzugsweise sind dies ganze Zahlen.

Ein binärer Baum wird ein *binärer Suchbaum* genannt, wenn er folgende Eigenschaften besitzt:

(1) Jeder Knoteninhalt besitzt als Komponente einen Schlüssel, für dessen Typ eine Ordnung definiert ist.
(2) Für jeden Knoten *K*, der kein Blatt ist, gilt: alle Schlüssel in den Knoten des linken Teilbaumes von *K* sind kleiner als der Schlüssel in *K*, alle Schlüssel in den Knoten des rechten Teilbaumes von *K* sind größer als der Schlüssel in *K*.

Der folgende Baum ist ein binärer Suchbaum mit ganzen Zahlen als Schlüsseln.



Wir verfolgen zunächst den Ablauf einer Suche in diesem Baum nach dem Schlüssel 41.

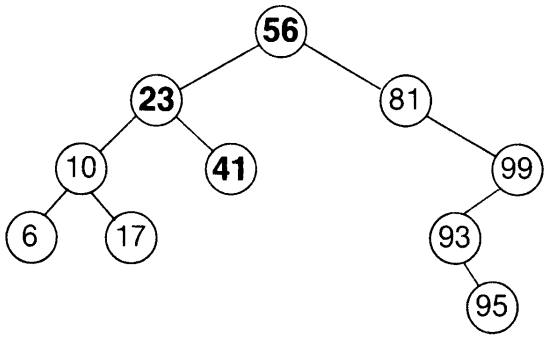
(1) Man vergleicht 41 mit dem Schlüssel 56 im Wurzelknoten. Es ist $41 <> 56$. Da $41 < 56$, sucht man im linken Teilbaum weiter.

(2) Man vergleicht 41 mit dem Schlüssel 23 in dessen Wurzelknoten. Es ist $41 <> 23$. Da $41 > 23$, sucht man im rechten Teilbaum weiter.

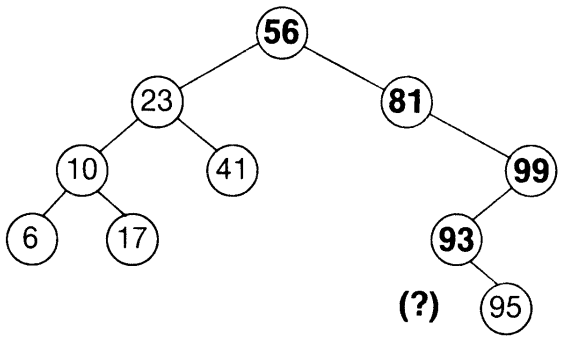
(3) Man vergleicht 41 mit dem Schlüssel 41 in dessen Wurzelknoten. Es ist $41 = 41$. Die Suche ist erfolgreich beendet.

Sucht man in ähnlicher Weise nach dem Schlüssel 85, so besucht man nacheinander die Knoten mit den Schlüsseln 56, 81, 99 und 93. Dort müßte man im linken Teilbaum weitersuchen. Dieser ist aber leer. Also ist die gesuchte Zahl nicht im Baum enthalten.

(a) Suche nach 41



(b) Suche nach 85



Wie die Beispiele zeigen, kann entsprechend der Definition eines binären Baumes der Suchalgorithmus rekursiv formuliert werden.

Algorithmus Suche Schlüssel X im binären Suchbaum b

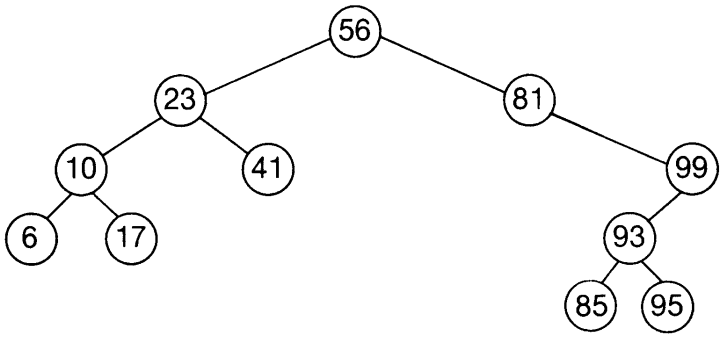
- Falls *b* leer ist, dann ist die Suche erfolglos.
- anderenfalls
  - Falls *X* gleich dem Schlüssel im Wurzelknoten ist, dann ist die Suche erfolgreich
  - anderenfalls
    - Falls *X* kleiner als der Schlüssel im Wurzelknoten ist, dann
      - Suche Schlüssel *X* im linken Teilbaum von *b*.
    - anderenfalls
      - Suche Schlüssel *X* im rechten Teilbaum von *b*.

Der Vorteil der Suche im binären Suchbaum gegenüber einer sequentiellen Suche im Feld oder in einer linearen Liste wird deutlich erkennbar. Mit jedem Suchschritt, der nicht bereits das Element findet, wird ein ganzer Teilbaum von der Weitersuche ausgeschlossen.

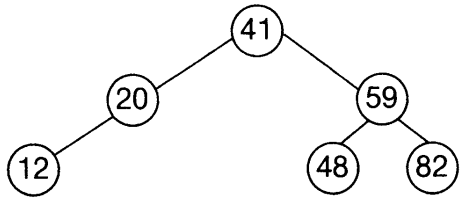
Der *Aufbau* eines binären Suchbaumes geschieht schrittweise. Beginnend mit einem leeren Baum werden nacheinander neue Element eingefügt. Jedes neue Element muß gezielt so an einen Platz gebracht werden, daß die kennzeichnende Eigenschaft (2) bestehen bleibt. Daher ist es nicht sinnvoll, dieses Einfügen einer äußeren Steuerung zu überlassen, wie dies z.B. beim Beruferaten der Fall war.

Dem *Einfügen* geht stets eine Suche voraus. Ein Element, dessen Schlüssel bereits vorhanden ist, wird nicht eingefügt. Ein nicht vorhandenes Element wird an der Stelle eingefügt, wo die Suche erfolglos abgebrochen wurde. Das bedeutet: der binäre Suchbaum erhält ein neues Blatt, dessen Inhalt das einzufügende Element ist.

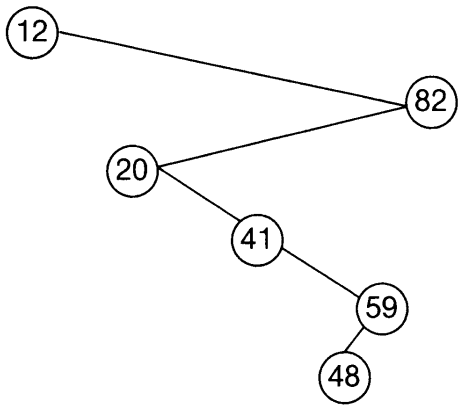
Für das obige Beispiel (Suche nach 85) heißt dies: die Zahl 85 müßte im linken Teilbaum von 93 stehen. Da dieser Teilbaum leer ist, wird ein neuer, einelementiger Binärbaum geschaffen, dessen Wurzelknoten den Inhalt 85 hat. Er wird als linker Teilbaum an den Knoten mit der 93 angehängt. Der Baum hat danach den folgenden Aufbau.



Dieses Verfahren hat den *Vorteil*, daß der Baum insgesamt nicht umorganisiert werden muß (wie dies etwa beim Einfügen in einem sortierten Feld der Fall ist). Ein *Nachteil* besteht aber darin, daß die Reihenfolge, in der man die einzelnen Elemente zum Aufbau des Baumes einfügt, die Gestalt des Baumes sehr stark beeinflussen kann. So ergeben einerseits die beiden Reihenfolgen 41 20 12 59 82 48 und 41 59 48 20 82 12 denselben binären Suchbaum



Dagegen entsteht aus der anderen Reihenfolge 12 82 20 41 59 48 derselben Zahlen der Baum



Hier besitzen die Wurzel und jeder innere Knoten jeweils genau einen nicht-leeren nachfolgenden Teilbaum. Diese Verknüpfung entspricht der einer linearen Liste. Man spricht von einem „entarteten" Suchbaum. Das Suchen in einem entarteten Suchbaum hat keine Vorteile mehr gegenüber der sequentiellen Suche in einer linearen Liste. In der überwiegenden Zahl der Anwendungsfälle ist die Reihenfolge der neu einzufügenden Elemente problembedingt und erzeugt im allgemeinen keinen entarteten Baum.

Bei vielen Anwendungen tritt auch die Notwendigkeit auf, einen Knoten in einem binären Suchbaum zu *löschen*. Ein Entfernen eines Knotens ist meist nicht so einfach wie das Einfügen, da man sich die Stelle nicht aussuchen kann. Man kann drei Fälle unterscheiden:

(i) Wenn der zu löschende Knoten ein Blatt ist, so kann er unmittelbar entfernt werden.

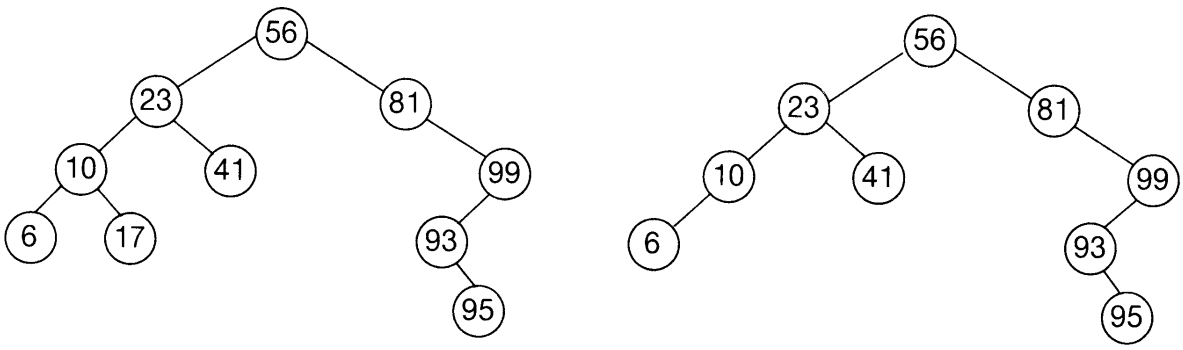
(ii) Hat er nur einen nichtleeren nachfolgenden Teilbaum, so kann er ähnlich wie in einer linearen Liste „überbrückt" und dann gelöscht werden.

(iii) Hat er dagegen zwei nichtleere Nachfolger, so darf der Knoten an dieser Stelle nicht entfernt werden. Denn sein Vorgänger kann nicht beide Nachfolger des gelöschten Knotens an dessen Stelle als Nachfolger erhalten. Eine Lösungsmöglichkeit besteht darin, gezielt einen Knoten zu suchen, der entweder ein Blatt ist oder nur einen Nachfolger hat. Dann wird dessen Inhalt in denjenigen Knoten übertragen, der den zu löschenden Inhalt besitzt. Schließlich wird dieser letztere Knoten aus dem Baum entfernt.

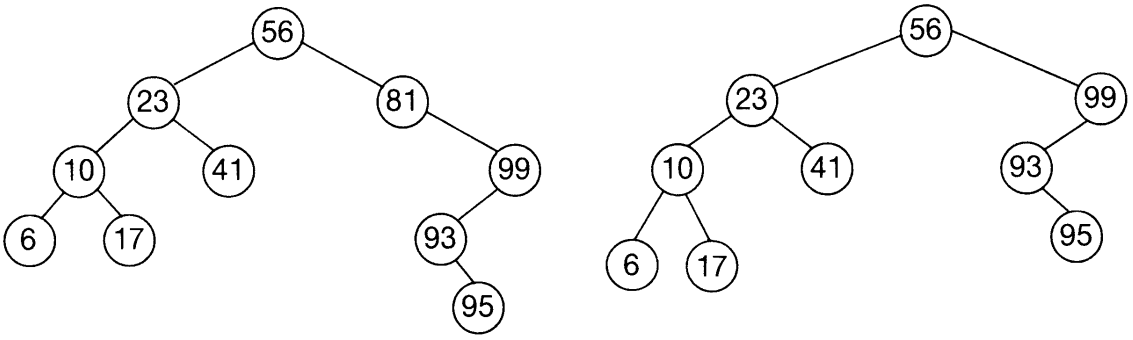
    Bei der Auswahl dieses Knotens muß man darauf achten, daß der Baum ein binärer Suchbaum bleibt. Man muß daher im linken nachfolgenden Teilbaum des zu löschenden Elementes, der ja nur kleinere Schlüssel enthält, den größten Wert suchen (oder im rechten Teilbaum den kleinsten). Dieser Wert ist größer als alle anderen Schlüssel in diesem linken Teilbaum, andererseits kleiner als alle Schlüssel im rechten Teilbaum des zu löschenden Wertes. Daher kann er an die Stelle des zu löschenden Schlüssels gesetzt werden.

Beispiele

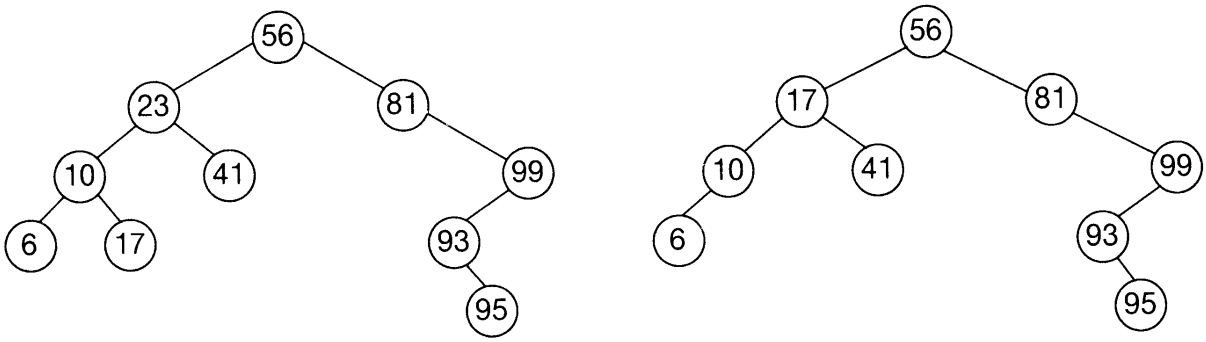
Zu (i): Löschen des Knotens mit dem Inhalt 17.
Man sucht zunächst den Knoten mit Inhalt 17 und durchläuft dabei die Knoten mit den Werten 56, 23, 10, 17. Da dieser Knoten ein Blatt ist, wird er aus dem Baum entfernt.



Zu (ii): Löschen des Knotens mit dem Inhalt 81.
Hier werden die Knoten mit den Werten 56, 81 besucht. Da der gesuchte Knoten nur einen nicht-leeren nachfolgenden Teilbaum hat, rückt dieser an die Stelle des Baumes mit der Wurzel 81. Der Knoten mit dem Inhalt 81 kann aus dem Baum entfernt werden. Wieder sind die Zustände vor und nach dem Löschen dargestellt.



Zu (iii): Löschen des Knotens mit dem Inhalt 23.
Zunächst werden die Knoten mit den Inhalten 56 und 23 besucht. Der gesuchte Knoten hat zwei nicht-leere nachfolgende Teilbäume und darf daher nicht entfernt werden. Deshalb sucht man im linken nachfolgenden Teilbaum den am weitesten rechts stehenden Knoten, denn dieser enthält den größten Schlüssel in diesem Teilbaum und hat höchstens einen linken nachfolgenden Teilbaum. Es ist die Zahl 17. Diese wird in den Knoten mit der 23 übertragen und ersetzt dort diesen Wert. Schließlich wird der alte Knoten mit der 17 aus dem Baum entfernt.



Zwei Vorgänge müssen deutlich unterschieden werden: das *Löschen eines Inhalts* und das *Entfernen eines Knotens*. Löschen bedeutet, daß ein Baum entsteht, bei dem in keinem Knoten der genannte Inhalt vorkommt. Entfernen heißt dagegen, daß ein Baum entsteht, der einen bestimmten Knoten nicht mehr enthält. In den beiden ersten Fällen wurde jeweils der Knoten mit dem zu löschenden Inhalt entfernt und damit auch der Inhalt gelöscht; denn es gibt keinen zweiten Knoten mit demselben Inhalt. Im dritten Fall wird dagegen zwar der Inhalt gelöscht, es wird aber nicht der Knoten entfernt, der diesen Inhalt enthält. Vielmehr wird der Inhalt durch Überschreiben mit einem anderen Inhalt gelöscht. Entfernt wird aber der Knoten, aus dem dieser andere Inhalt kopiert wurde.

Algorithmus zum Löschen im binären Suchbaum

Handlungsanweisung:

- Suche den Knoten *K* mit dem zu löschenden Inhalt.
- Falls sein linker Teilbaum leer ist, dann
  - Hänge seinen (möglicherweise leeren) rechten Teilbaum an die Stelle des Baumes, dessen Wurzel *K* ist.

  anderenfalls
  - Falls sein rechter Teilbaum leer ist, dann
    - Hänge seinen linken Teilbaum an die Stelle des Baumes, dessen Wurzel *K* ist.

    anderenfalls
    - Laufe im linken Teilbaum soweit nach rechts, bis ein Knoten *K1* angetroffen wird, der keinen rechten Teilbaum hat.
    - Übertrage seinen Inhalt in den Knoten *K*.
    - Hänge den linken Teilbaum von *K1* an die Stelle des Baumes, dessen Wurzel *K1* ist.

Übungen

1. Gegeben ist die Zahlenfolge 20 7 63 5 3 76 17 1 38 19 23 9 67. Die Zahlen sollen in der vorgegebenen Reihenfolge in einen anfangs leeren binären Suchbaum eingefügt werden. Zeichnen Sie die den Baum, der dadurch entsteht.

2. Bauen Sie in ähnlicher Weise einen binären Suchbaum aus den Wörtern

   PROGRAM CONST TYPE VAR RECORD INTEGER BEGIN END AND IF THEN ELSE REPEAT UNTIL ARRAY.

3. Löschen Sie im Baum der Übung 1 nacheinander die Zahlen 23, 76, 7, 20. Zeichnen Sie nach jedem Löschvorgang den Baum neu.