Aufgabensammlung zum Buch „Algorithmen und Datenstrukturen“ (Kapitel 5)

1 Aufgaben aus dem Buch

Zu folgenden Aufgaben, die direkt aus dem Buch entnommen sind, gibt es an der Universität Freiburg am Lehrstuhl Ottmann Musterlösungen. In der Version mit Lösungen sind diese angegeben. Hinter der fortlaufenden Aufgabennummer steht in Klammern die Nummer der Aufgabe im Buch.

2 Ähnlich Aufgaben

Bei den folgenden Aufgaben handelt es sich um Aufgaben die an der ETH Zürich, am Institut für Theoretische Informatik und an der Universität Freiburg im Institut für Informatik in diversen Vorlesungen gestellt wurden. Inhaltlich sind diese Aufgaben mit dem behandelten Stoff im Buch verwandt. Zu allen Aufgaben gibt es Musterlösungen, die allerdings nur in der Version mit Lösungen enthalten sind.

Aufgabe 1:

In **binären Suchbäumen** werden Daten gespeichert, die mit Hilfe von Schlüsseln angeordnet werden können. In einem binären Suchbaum gilt für jeden Knoten p, dass die Schlüssel im linken Teilbaum von p kleiner und die im rechten Teilbaum grösser als der Schlüssel von p sind. Ein binärer Suchbaum stellt folgende Operationen zur Verfügung:

insert(T,x): Falls ein Objekt y im Baum T existiert, das denselben Schlüssel wie x hat, so wird y durch x ersetzt; andernfalls wird x in den Baum eingefügt.

delete(T,k): Falls ein Objekt x im Baum T existiert, dessen Schlüssel k ist, so wird x aus dem Baum entfernt; andernfalls bleibt der Baum unverändert.

lookup(T,k): Falls ein Objekt x im Baum T existiert, dessen Schlüssel k ist, so wird x zurückgegeben; andernfalls wird ”nichts” zurückgegeben.

Implementiere einen binären Suchbaum.

Aufgabe 2:

Bei einem *Preorder*-Durchlauf (Hauptreihenfolge) eines Binärbaumes mit Wurzel p wird zunächst p ausgegeben, dann die Knoten des linken Teilbaumes von p in *Preorder*-Reihenfolge und anschliessend die Knoten des rechten Teilbaumes von p in *Preorder*-Reihenfolge.

1. Implementiere eine Funktion, die die Knoten eines Binärbaumes, wie er in Aufgabe 4.1 realisiert wurde, in *Preorder*-Reihenfolge ausgibt.

2. Entwirf einen nicht-rekursiven Algorithmus, der mit Hilfe eines Stacks die Knoten eines Binärbaumes in *Preorder*-Reihenfolge ausgibt.

   Hinweis: Mit dem Stack aus Aufgabe 2.4 kannst du diesen Algorithmus natürlich auch implementieren.

Aufgabe 3:

Für einen binären Baum T definieren wir die Höhe rekursiv:

$$height(T) := \begin{cases} 0 & \text{falls } T \text{ genau einen Knoten enthält,} \\ 1 + max(height(T_l), height(T_r)) & \text{sonst.} \end{cases}$$

T_l und T_r sind dabei der linke bzw. rechte Teilbaum, der an der Wurzel von T hängt. Anders ausgedrückt ist die Höhe von T die Länge eines längsten Pfades von der Wurzel von T zu einem Blatt.

Implementiere eine Funktion, die für einen binären Baum, wie er in Aufgabe 4.1 realisiert wurde, die Höhe bestimmt.

Aufgabe 4:

Ein AVL-Baum ist ein binärer Suchbaum, in dem sich für jeden Knoten p die Höhe der beiden Teilbäume von p um höchstens 1 unterscheidet. Solche Bäume heissen höhenbalanciert.

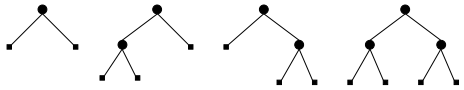
1. Gib den AVL-Baum an, der durch Einfügen der Schlüssel $1, 2, 1, 9, 12, 11, 9, 10, 3, 4, 5$ in einen leeren Baum entsteht.
2. Gibt es zu jedem AVL-Baum T eine Einfügereihenfolge der Schlüssel, die zu diesem Baum führt, **ohne** dass Rotationen stattfinden? Falls ja, so gib einen Beweis an, andernfalls ein Gegenbeispiel.

Aufgabe 5:

1. Zeige, daß ein binärer Baum mit n Knoten genau $n - 1$ Kanten enthält.
2. Zeige, daß ein binärer Baum mit n Knoten mindestens Höhe $\lceil \log(n + 1) \rceil - 1$ hat.
3. Ein binärer Baum heißt vollständig, wenn jeder innere Knoten genau zwei Kinder hat. Zeige, daß ein vollständiger binärer Baum mit n Blättern genau $n - 1$ innere Knoten hat.

Aufgabe 6:

In der Vorlesung wurden geschichtete Bäume vorgestellt, die auf folgender Menge von vier Grundbäumen beruhen:



Gib eine weitere Menge von Grundbäumen an, mit der ebenfalls geschichtete Bäume konstruiert werden können. Beschreibe skizzenhaft, wie beim Einfügen bzw. Löschen eines Elementes restrukturiert werden muss.

Aufgabe 7:

Für die vier Schlüssel F, I, N und O soll ein optimaler Suchbaum erstellt werden. Die Zugriffshäufigkeiten sind in folgender Tabelle angegeben:

$(-\infty, F)$	F	(F, I)	I	(I, N)	N	(N, O)	O	(O, ∞)
3	4	2	5	3	3	0	2	1

Auf den Wert I wird also fünf mal, auf einen Wert zwischen I und N wird drei mal zugegriffen.

Erzeuge von Hand einen optimalen Suchbaum für diese vier Schlüssel.

Aufgabe 8:

Wir wollen drei verschiedene binäre Suchbäume für die 31 häufigsten englischen Begriffe erzeugen und vergleichen. Die Begriffe findest Du in der folgenden Tabelle:

a	5074	for	1869	in	4312	the	15568
and	7638	from	1039	is	2509	this	1021
are	1222	had	1062	it	2255	to	5739
as	1853	have	1344	not	1496	was	1761
at	1053	he	1727	of	9767	which	1291
be	1535	her	1093	on	1155	with	1849
but	1379	his	1732	or	1101	you	1336
by	1392	i	2292	that	3017		

Sei

$$P := \sum_{i=1}^{N} (Tiefe(k_i) + 1) a_i$$

die gewichtete Pfadlänge eines Baumes[1]. Dabei ist $N = 31$ die Anzahl der Schlüssel, k_i der i-te Schlüssel und a_i die Zugriffshäufigkeit des i-ten Schlüssels.

1. Erzeuge einen binären Suchbaum, indem du die 31 Begriffe zunächst gemäss ihren Häufigkeiten **absteigend** sortierst und sie dann in dieser Reihenfolge in einen leeren binären Suchbaum einfügst.
2. Erzeuge einen AVL-Baum, indem du die Wörter gemäss ihren Häufigkeiten **absteigend** sortierst und sie dann in dieser Reihenfolge in einen leeren AVL-Baum einfügst.
3. Erzeuge einen optimalen Suchbaum für die 31 Begriffe.

Gib für jeden der drei Suchbäume die gewichtete Pfadlänge an.

Aufgabe 9:

In Abbildung 1 ist ein binärer Suchbaum mit 13 Schlüsseln angegeben. In diesem Suchbaum wird zunächst auf den Schlüssel 50 und anschliessend auf den Schlüssel 23 zugegriffen.

1. Gib den Baum an, der durch die beiden Zugriffe entsteht, wenn die Move-To-Root-Strategie angewendet wird.
2. Wie sieht der Baum nach den Zugriffen aus, wenn es sich um einen Splay-Tree handelt?

Aufgabe 10:

Sortieren Sie mit natürlichen Suchbäumen!

1. Erzeugen Sie algorithmisch alle Permutationen von $[1, 2, 3, 4, 5]$. Nutzen Sie zur Aufzählung eine rekursive Prozedur `permute` anstatt fünf ineinandergeschachtelte Schleifen. Wieviel Vertauschungen benötigen Sie?

[1] Der Term $\sum_{j=0}^{N} Tiefe(Blatt(k_j, k_{j+1}))b_j$, der im Buch vorkommt, taucht hier nicht auf, da die Häufigkeiten für erfolgloses Suchen nicht bekannt sind und wir alle $b_j := 0$ setzen.

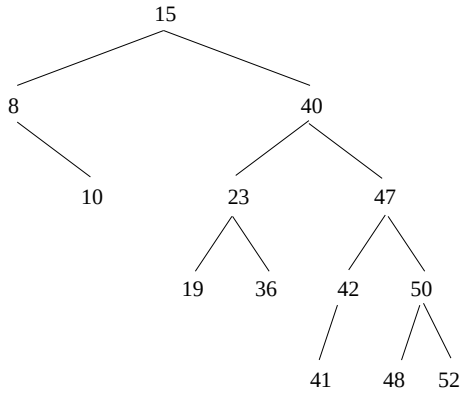


Abbildung 1: Binärer Suchbaum mit $n = 13$ Schlüsseln

2. Gegeben sei eine Menge von n vergleichbaren (o.B.d.A. paarweise verschiedenen) Arrayelementen `a[0]` bis `a[n-1]`. Zu Vereinfachung beinhalte `a` eine Permutationen von $[1, 2, 3, 4, 5]$. Fügen Sie diese Elemente in einen natürlichen Suchbaum ein und durchlaufen Sie den Baum in der symmetrischen Reihenfolge. Sie erhalten ein einfaches Sortierverfahren.

   Wie gut ist das Verfahren im besten, mittleren und schlechtesten Fall? Beantworten Sie diese Frage experimentell, indem Sie die Suchbaumimplementation aus der Vorlesung nutzen. Mitteln Sie über alle Permutationen der Elemente in `a`.

Aufgabe 11:

Sei $T(r)$ ein geordneter Baum (die Kinder eines Knotens sind linear geordnet) mit Wurzel r und Kindern $v_1, \ldots, v_n$.

Dann ist der Postorderdurchlauf $post(T)$ definiert als $post(T(v_1)) \ldots post(T(v_k))\ r$. Entsprechend sind der Inorderdurchlauf $in(T)$ als $in(T(v_1))\ r\ in(T(v_2)) \ldots in(T(v_k))$, der Preorderdurchlauf $pre(T)$ als $r\ pre(T(v_1)) \ldots pre(T(v_k))$ und der Levelorderdurchlauf $lev(T)$ als $r\ \ lev(T(v_1))\ \ r\ \ lev(T(v_2)); \ldots$ $r\ lev(T(v_k))\ r$ festgelegt.

Beweisen Sie die folgenden Aussagen für paarweise verschiedene Schlüssel in T oder finden Sie ein Gegenbeispiel.

1. Aus der Levelorder kann T eindeutig rekonstruiert werden.
2. Preorder und Inorder beschreiben einen Baum T eindeutig.
   Unterscheiden Sie dabei den binären (max. zwei Kinder) und den allgemeinen Fall.
3. Preorder und Postorder beschreiben T eindeutig.

Aufgabe 12:

Fügen Sie die Monatsnamen nacheinander in einen lexikographisch geordneten, anfangs leeren AVL-Baum ein. Wenn immer eine Rebalancierung notwendig ist, zeichnen Sie den Baum.

Aufgabe 13:

Betrachten Sie die Struktur eines AVL-Baumes aus Abbildung 1.

1. Geben Sie die verschiedenen Strukturen der AVL-Bäume an, die durch die Einfügung von einem Schlüssel entstehen können.
2. Vorausgesetzt, daß die Schlüssel mit der gleichen Wahrscheinlichkeit in jedes der Intervalle zwischen den bereits im Baum befindlichen Schlüsseln fallen; wie groß ist die Wahrscheinlichkeit, daß die Höhe des angegebenen AVL-Baumes nach zwei Einfügungen um eins wächst?

Aufgabe 14:

(Knotengefärbte) Rotschwarz-Bäume sind binäre Suchbäume mit den folgenden Eigenschaften:

i) Jeder Knoten ist entweder rot oder schwarz.

ii) Jedes Blatt ist schwarz.

iii) Falls ein Knoten rot ist, dann sind beide Kinder schwarz.

iv) Jeder Pfad von einem Knoten zu einem Blatt enthält die gleiche Anzahl von schwarzen Knoten.

Zeigen Sie

1. Ein (knotengefärbter) Rotschwarzbaum mit n inneren Knoten besitzt eine Höhe H von höchstens $2\log(n+1)$.
2. Entsprechend den AVL-Bäumen gibt es Operationen `Left-Rotate` und `Right-Rotate`. Skizzieren Sie die Einfärbungs- und Balancierungsoperationen der Einfügeprozedur eines Elementes in einem Rotschwarzbaum in Pseudocode. Dabei starten Sie mit dem eingefügten Knoten und der Färbung "rot" und hangeln sich je nach Lage und Farbe der Ahnen mittels Rotationen und Umfärbungen über die Vorgängerverweise an den Knoten zu den Eltern und Ureltern hinauf.

Aufgabe 15:

Persistente Datenstrukturen ermöglichen einen Zugriff in die Vergangenheit der dynamischen Veränderungen. Wir konzentrieren uns auf persistente Suchbäume. Eine Möglichkeit zur Implementation ist es, eine Kopie des Suchbaumes anzulegen, wenn immer er sich verändert.

Eine effizientere Möglichkeit (vergleiche Abbildung 2) besteht darin, die sich nicht verändernden Teile der Datenstruktur wiederzuverwerten.

1. Beschreiben Sie die Knoten, die bei der Einfügung eines Schlüssels k oder bei der Löschung eines Knotens y verändert werden müssen.

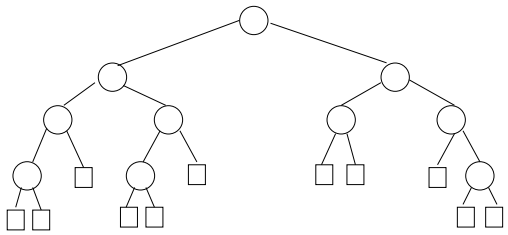


Abbildung 2: Die Struktur eines AVL-Baumes mit 10 Schlüsseln.

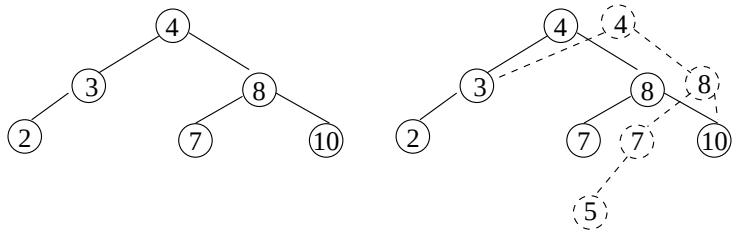


Abbildung 3: Ein persistenter Suchbaum vor und nach dem Einfügen von 5.

2. Angenommen, die derzeitige Höhe des Suchbaumes wäre durch h gegeben und Knoteninformation bestünde aus dem Schlüssel, dem linken und dem rechten Nachfolger. Was ist die Laufzeit des Einfügens eines Knotens in die persistente Suchbaumstruktur?

3. Geben Sie Argumente dafür an, daß sich der Algorithmus bei gleicher Laufzeit auf balancierte Suchbäume erweitern läßt.

Aufgabe 16:

Wörterbücher bieten die Operation des Suchens, des Einfügens, und des Löschens eines Datensatzes aufgrund seines Schlüssels an. AVL-Bäume führen diese Operationen in logarithmischer Zeit zur Anzahl der Datenelemente durch. Wenn wir ganzzahlige Schlüssel aus dem Intervall $[0 \ldots m]$ ablegen wollen, so können wir diese Operationen durch ein Array (Schubladen) von AVL-Bäumen $A_0, \ldots, A_{k-1}$ noch weiter beschleunigen. Hierbei wird der Schlüssel i in den AVL-Baum mit Nummer $(i \bmod k)$ gesucht, eingefügt oder gelöscht.

Implementieren sie die Klasse `SchubladenAVL` mit den Methoden `insert`, `delete`, `search` und `print`. Als Test wähle man $k = 7$, füge die Zahlen von 0 bis 90 ein und lösche die Zahlen von 21 bis 69 gemäß dem folgenden Schema.

```
public class SchubladenTest {
   public static void main(String args[]) {
      System.out.println ("SchubladenTest");
      System.out.println ("Erzeuge 7 Schubladen von AVL-Baeumen");
      SchubladenAVL T = new SchubladenAVL (7);
      System.out.println ("Fuege Zahlen 0 .. 90 ein:");
      for (int i=0; i<=90; ) T.insert (i++);
      System.out.println ("Loesche Zahlen 69 .. 21:");
      for (int i=69; i>=21; ) T.delete (i--);
      System.out.println ("Ist 90 enthalten? "+T.search(90));
      System.out.println ("Ist 21 enthalten? "+T.search(21));
   }
}
```

Aufgabe 17:

Implementieren Sie einen Bruderbaum inklusive der beiden Wörterbuch-Operationen *Einfügen* und *Suchen*.

Wählen Sie die Schlüsselmenge $\{1, 2, 3, 4, 5\}$ als Testbeispiel. Die (textuelle) Ausgabe sollte der Abbildung 5.28 im Buch Ottmann/Widmayer (3. Auflage) entsprechen.

Aufgabe 18:

Zeichnen Sie für die Schlüsselmenge $S = \{1, 2, 3\}$ alle Suchbäume der Höhe 3.

Aufgabe 19:

Sei $N = 2^h - 1$ mit h als natürliche Zahl. Sei $T(h)$ die Anzahl der Permutationen einer N-elementigen Schlüsselmenge, die beim Einfügen in den anfangs leeren Baum einen vollständigen Suchbaum erzeugen. Geben Sie ein Rekursionsgleichungssystem zur Bestimmung von $T(h)$ an. Berechnen Sie danach $T(h)$ für $h \in \{0, 1, 2, 3\}$.

Aufgabe 20:

Wieviele verschiedene Strukturen $S(N)$ von Suchbäumen für N Schlüssel der Höhe N gibt es? Geben Sie eine kurze Begründung an!

Aufgabe 21:

Sei $T(r)$ ein Baum mit Wurzel r und Kindern $v_1, \ldots, v_n$. Dann ist der Postorderdurchlauf $post(T)$ definiert als $post(T(v_1)) \ldots post(T(v_k))\ r$. Entsprechend sind der Inorderdurchlauf $in(T)$ als $in(T(v_1))\ r\ in(T(v_2)) \ldots in(T(v_k))$, der Preorderdurchlauf $pre(T)$ als $r\ pre(T(v_1)) \ldots pre(T(v_k))$ und der Levelorderdurchlauf $lev(T)$ als $r\ lev(T(v_1))\ r\ lev(T(v_2))\ \ldots\ r\ lev(T(v_k))\ r$ festgelegt. Beweisen Sie die folgenden Aussagen für paarweise verschiedene Schlüssel in T oder finden Sie ein Gegenbeispiel.

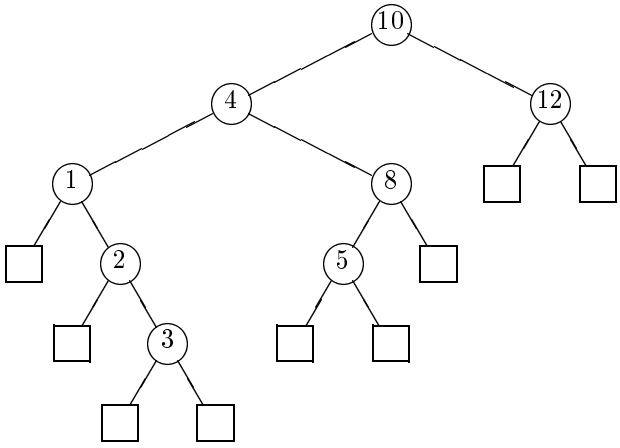
1. Aus der Levelorder kann T eindeutig rekonstruiert werden.
2. Preorder und Inorder beschreiben einen Baum T eindeutig.
   Unterscheiden Sie dabei den binären (max. zwei Kinder) und den allgemeinen Fall.
3. Preorder und Postorder beschreiben T eindeutig.

Aufgabe 22:

Zeigen Sie: Ein k-närer Baum (jeder innere Knoten hat k Söhne) mit r Blättern hat insgesamt (Blätter und innere Knoten) $r + \frac{r-1}{k-1}$ Knoten.

Aufgabe 23:

Es sei der folgende natürliche Suchbaum t gegeben.



1. Geben Sie für diesen Baum die interne Pfadlänge an?
2. Was ist die durchschnittliche interne Pfadlänge von t?
3. Führen Sie in dem unten angegebenen Binärsuchbaum der Reihe nach die folgenden Operationen durch:

   einfügen(11), einfügen(6), entferne(12), entferne(4).

Aufgabe 24:

1. Fügen Sie die Schlüssel 28,7,69,6,20,10,4,2 in einen (anfänglich leeren) AVL-Baum ein. Geben Sie neben dem Endergebnis mit entsprechenden Balance-Werten die Anzahl der Rotationen pro Schlüssel an. Wieviele Rotationen kann es beim Einfügen in einen AVL-Baum maximal geben?
2. Fügen Sie die Schlüssel 28,7,69,6,20,10,4,2 in einen (anfänglich leeren) Bruder-Baum ein. Geben Sie neben den Endergebnis die Anzahl der Aufrufe *Up* pro Schlüssel an. Wie groß ist der mittlere Aufwand zum Einfügen eines Schlüssels (O-Notation).

Aufgabe 25:

Analog zur Familie $\{B_k\}$ der Binomial Trees läßt sich eine Familie $\{T_k\}$ von Bäumen aufbauen, die man *Trinomial Trees* nennen könnte. Die Familie soll folgende Eigenschaften erfüllen:

1. Die Anzahl der Knoten in T_i ist 3^i.
2. Die Wurzel von T_i hat Grad $2i$.
3. Die Höhe von T_i ist i.

Versuchen Sie, die Struktur der Trinomial Trees zu beschreiben:

a) Definieren Sie $\{T_k\}$ und zeigen Sie, daß die genannten Eigenschaften gelten.

b) Erläutern Sie, wie sich mit den Trinomial Trees auch *Trinomial Heaps* definieren lassen. Welche Veränderungen ergeben sich gegenüber Binomial Heaps? Wie wird die Operation *Merge*, also die Vereinigung zweier Trinomial Trees, durchgeführt?