Datenstrukturen und Algorithmen, SS 2019
Prof. Dr. P. Rossmanith
S. Dollase, J. Dreier, H. Lotze

Datum: 25.04.2019

Übungsblatt mit Lösungen 03

Aufgabe T9

a) Fügen Sie die Knoten 1 bis n in aufsteigender Reihenfolge in einen Binärbaum ein. Was ist die Laufzeit?

b) Löschen Sie nun diese Knoten in der gleichen Reihenfolge. Was ist die Laufzeit?

Lösungsvorschlag

a) Angenommen es wurden bereits die Knoten 1 bis i eingefügt. Der Knoten $i+1$ wird als rechtes Kind unter dem Knoten i eingefügt. Dies dauert Zeit $O(i)$. Die Gesamtlaufzeit ist

$$\sum_{i=1}^{n} O(i) = O(\sum_{i=1}^{n} i) = O(n(n-1)/2) = O(n^2).$$

b) Angenommen es wurden bereits die Knoten 1 bis i gelöscht. Die aktuelle Wurzel ist der Knoten $i+1$. Diesen zu löschen dauert konstante Zeit. Die Gesamtlaufzeit ist

$$\sum_{i=1}^{n} O(1) = O(n).$$

Aufgabe T10

Wir fangen mit einem leeren AVL-Baum an. Führen Sie die Schritte a) bis e) nacheinander aus und zeichnen Sie den Binärbaum nach jedem Schritt.

a) Fügen Sie die Zahlen 1,...,5 in dieser Reihenfolge ein.

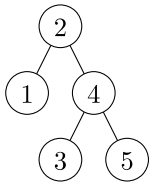
b) Löschen Sie die 2.

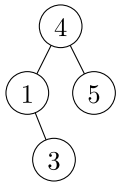
c) Fügen Sie die 2 ein.

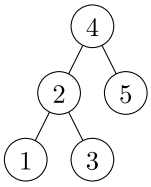
d) Löschen Sie die 2.

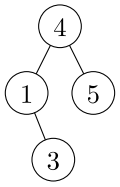
e) Fügen Sie die 2 ein.

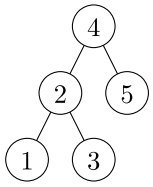
Lösungsvorschlag

a) 

b) 

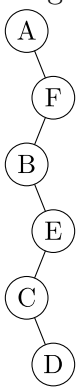
c) 

d) 

e) 

Aufgabe T11

Ist es möglich, dass ein optimaler Suchbaum folgende Form annimmt?



Falls ja: Geben Sie eine mögliche Verteilung der Zugriffswahrscheinlichkeiten auf die Knoten A – F an. Falls nicht, argumentieren Sie (informell) wieso dies nicht möglich sein kann.

Lösungsvorschlag

Dies ist tatsächlich möglich. Dabei müssen Zugriffswahrscheinlichkeiten der Knoten so verteilt werden, dass diese absteigend in den Baum relativ zueinander immer geringer werden. Wir müssen dabei selbstverständlich darauf achten, dass die Summe der Wahrscheinlichkeiten weiterhin 1 ergibt. Wir können dies einfach sicherstellen, in dem wir die Wahrscheinlichkeiten zum Beispiel wie folgt auf die Knoten (von oben nach unten) für ein kleines ε verteilen:

1. $1 - \varepsilon$
2. $\varepsilon - \varepsilon^2$
3. $\varepsilon^2 - \varepsilon^3$
4. $\varepsilon^3 - \varepsilon^4$
5. $\varepsilon^4 - \varepsilon^5$
6. ε^5

Die konkrete Verteilung auf die Knoten lautet also: A$(1 - \varepsilon)$, B$(\varepsilon^2 - \varepsilon^3)$, C$(\varepsilon^4 - \varepsilon^5)$, D$(\varepsilon^5)$, E$(\varepsilon^3 - \varepsilon^4)$, F$(\varepsilon - \varepsilon^2)$

Wählen wir zum Beispiel $\epsilon = 0.01$. Zunächst argumentieren wir, dass bei einem optimalen Suchbaum die Wurzel immer A ist. Falls die Wurzel A ist, so ist die erwartete Anzahl an Vergleichen maximal $1 \cdot (1-\varepsilon) + 6 \cdot \varepsilon$. In dieser Abschätzung wurde verwendet, dass für jeden Knoten maximal 6 Vergleiche notwendig sind. Wäre die Wurzel nicht A, so braucht man mindestens $2 \cdot (1 - \varepsilon)$ Vergleiche. Für kleines ε gilt $1 \cdot (1 - \varepsilon) + 6 \cdot \varepsilon \leq 2 \cdot (1 - \varepsilon)$. Somit ist A die Wurzel.

Von nun an nehmen wir an, dass die Wurzel A ist. Als nächstes argumentieren wir, dass B auf Level zwei sein muss. Falls B auf Level zwei ist, so ist die erwartete Anzahl an Vergleichen maximal $1 \cdot (1 - \varepsilon) + 2 \cdot (\varepsilon - \varepsilon^2) + 6\varepsilon^2$.

Wäre B nicht auf Level 2, so bräuchte man mindestens $1 \cdot (1 - \varepsilon) + 3 \cdot (\varepsilon - \varepsilon^2)$ Vergleiche. Für kleines ε gilt $2 \cdot (\varepsilon - \varepsilon^2) + 6\varepsilon^2 \leq 3 \cdot (\varepsilon - \varepsilon^2)$. Somit ist B auf Level zwei. Setzt man diese Argumentation fort, so sieht man, dass der angegebene Baum tatsächlich ein Optimaler Suchbaum ist.

Stattdessen hätte man auch das Verfahren aus der Vorlesung verwenden können, um aus einer gegebenen Wahrscheinlichkeitsverteilung einen optimalen Suchbaum zu bauen.

Aufgabe H7 (10 Punkte)
Aus der Vorlesung kennen wir bereits die binäre Suche in einem sortierten Array. In dieser Aufgabe wollen wir in einem ggf. zyklisch verschobenen sortierten Array ohne Duplikate die binäre Suche durchführen. Es darf also keine Zahl mehrfach im Array enthalten sein.
Zum Beispiel wird aus dem sortierten Array

$$[1, 4, 8, 12, 15, 18, 23, 25, 36]$$

durch zyklisches verschieben um drei Stellen das folgende zyklisch verschobene sortierte Array:

$$[23, 25, 36, 1, 4, 8, 12, 15, 18]$$

a) Implementieren Sie die Methode `search`, welche durch binäre Suche in einem sortierten, jedoch ggf. zyklisch verschobenen Array ein gegebenes Element findet und den entsprechenden Index zurückgibt. Dabei ist es nicht notwendig programmatisch zu überprüfen ob das Array tatsächlich die erwartete Form hat.

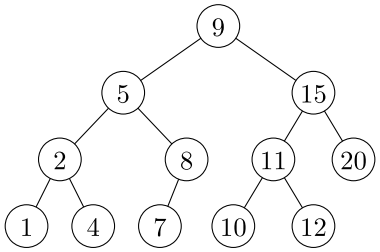
b) Welche Laufzeit hat ihre Implementierung? Begründen Sie Ihre Antwort.

Lösungsvorschlag

a)
```
public int search(int[] a, int element) {
    int left = 0;
    int right = a.length - 1;
    while (true) {
        int mid = (left + right) / 2;
        if (a[mid] == element) {           // found element
            return mid;                    // -> return index
        } else if (left == right) {        // one element remaining
            return -1;                     // -> return -1
        } else if (a[left] < a[right]) {   // remaining array is sorted
            if (element < a[mid]) {        // element to the left?
                right = mid - 1;           // -> search left half
            } else if (a[mid] < element) { // element to the right?
                left = mid + 1;            // -> search right half
            }
        } else if (a[left] < a[mid]) {     // left half is sorted ...
            if (a[left] <= element && element < a[mid]) {
                                           // ... and should contain element
                right = mid - 1;           // -> search left half
            } else {                       // ... and cannot contain element
                left = mid + 1;            // -> search right half
            }
        } else if (a[mid] < a[right]) {    // right half is sorted ...
            if (a[mid] < element && element <= a[right]) {
                                           // ... and should contain element
                left = mid + 1;            // -> search right half
            } else {                       // ... and cannot contain element
                right = mid - 1;           // -> search left half
            }
        }
    }
}
```

b) Das Program hat die Laufzeit $O(\log(n))$, da der Suchraum zu beginn die Größe n hat und anschließend mit jedem Schleifendurchlauf halbiert wird.

Aufgabe H8 (10 Punkte)
Gegeben ist folgender AVL-Baum:



Wenden Sie im folgenden Operationen immer auf den entstandenen Baum der vorherigen Teilaufgabe an.

a) In welcher Reihenfolge könnten die Schlüssel eingefügt worden sein, so dass gerade dieser AVL-Baum entsteht?

b) Wie sieht der Baum aus, wenn wir die 13 einfügen?

c) Was erhalten wir, wenn 7 und 8 gelöscht werden?

d) Jetzt wird die 9 gelöscht. Wie sieht der Baum danach aus?

e) Zuletzt fügen wir 14 ein. Was erhalten wir dadurch?

Lösungsvorschlag

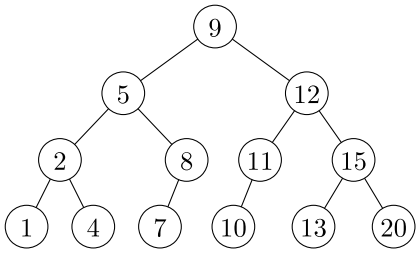
a) Am einfachsten ist es einzusehen, dass die Schlüssel „schichtenweise“ von oben nach unten eingefügt worden sein können. Dabei finden gar keine Rotationen statt, da der Baum immer balanziert bleibt. Diese Reihenfolge ist, wenn wir von links nach rechts gehen:

$$9, 5, 15, 2, 8, 11, 20, 1, 4, 7, 10, 12$$

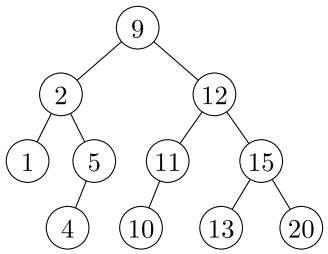
Es gibt natürlich viele andere Möglichkeiten. Allein wenn wir schichtenweise vorgehen, gibt es $2! \cdot 4! \cdot 5! = 5760$ Möglichkeiten.

Insgesamt gibt es aber noch viel mehr Reihenfolgen, nämlich genau 7,076,160.

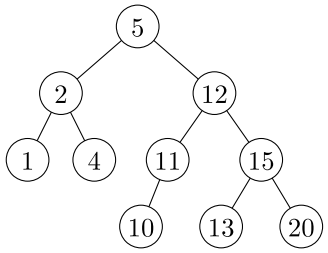
b)



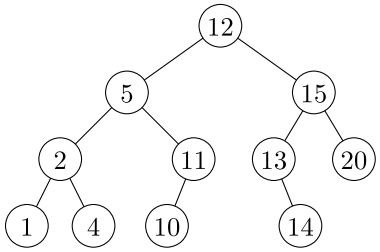
c)



d)



e)



Aufgabe H9 (10 Punkte)

Konstruieren Sie einen optimalen Suchbaum bezüglich der lexikographischen Ordnung für folgende Suchwörter auf die mit den gegebenen Wahrscheinlichkeiten zugegriffen wird: Hase(0.35), Hund(0.25), Katze(0.05), Maus(0.10), Pferd(0.20), Vogel(0.05).

Erstellen Sie dazu außerdem die Tabellen für $w_{i,j}$ und $e_{i,j}$.

Lösungsvorschlag

$w_{i,j}$	Hase	Hund	Katze	Maus	Pferd	Vogel
Hase	0.35	0.60	0.65	0.75	0.95	1.00
Hund	0.00	0.25	0.30	0.40	0.60	0.65
Katze	0.00	0.00	0.05	0.15	0.35	0.40
Maus	0.00	0.00	0.00	0.10	0.30	0.35
Pferd	0.00	0.00	0.00	0.00	0.20	0.25
Vogel	0.00	0.00	0.00	0.00	0.00	0.05

$e_{i,j}$	Hase	Hund	Katze	Maus	Pferd	Vogel
Hase	0.35(Hase)	0.85(Hase)	1.00(Hase)	1.30(Hund)	1.85(Hund)	2.00(Hund)
Hund	0.00	0.25(Hund)	0.35(Hund)	0.60(Hund)	1.15(Hund)	1.30(Maus)
Katze	0.00	0.00	0.05(Katze)	0.20(Maus)	0.55(Pferd)	0.65(Pferd)
Maus	0.00	0.00	0.00	0.10(Maus)	0.40(Pferd)	0.50(Pferd)
Pferd	0.00	0.00	0.00	0.00	0.20(Pferd)	0.30(Pferd)
Vogel	0.00	0.00	0.00	0.00	0.00	0.05(Vogel)