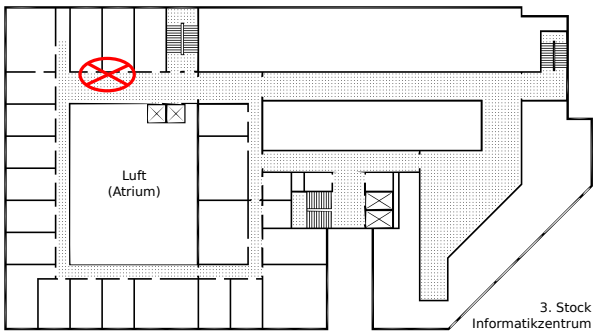
Abteilung Algorithmik **Winter 2015/16**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Dr. Christian Scheffer
Jan-Marc Reinhardt

Algorithmen und Datenstrukturen
Übung 4 vom 06. 01. 2016

Abgabe der Lösungen bis zum Mittwoch, den 20. 01. 2016 um 11:15 im Hausaufgabenrückgabeschrank.

Bitte die Blätter zusammenheften und vorne deutlich mit eigenem Namen, Matrikel- und Gruppennummer, sowie Studiengang versehen!



Aufgabe 1 (Binäre Suchbäume): Wir betrachten binäre Suchbäume (keine AVL-Bäume).

a) Füge nacheinander die folgenden Elemente in einen zu Beginn leeren binären Suchbaum ein. Gib den Baum nach jeder Einfügeoperation an:

$$4, \quad 11, \quad 7, \quad 5, \quad 8, \quad 15, \quad 1$$

b) Lösche die 11 aus dem konstruierten Baum. Beschreibe kurz, wie du dabei vorgehst. Gib den Baum nach dem Löschen an.

c) Ist der nach Aufgabenteil b) entstandene Baum ein AVL-Baum? Begründe deine Antwort.

(14+4+2 Punkte)

Aufgabe 2 (AVL-Bäume): Gegeben ist der AVL-Baum T aus Abbildung 1. Führe nacheinander die unten aufgeführten Operationen aus und gib T nach jedem Hinzufügen, Löschen oder Restrukturieren eines Knotens an. Falls nach einer INSERT oder DELETE Operation kein RESTRUCTURE notwendig ist, gib für jeden Knoten die Differenz der Höhen des linken und rechten Teilbaumes an.

a) INSERT$(T, 23)$

b) DELETE$(T, 5)$

c) DELETE$(T, 10)$

d) DELETE$(T, 17)$

e) INSERT$(T, 24)$

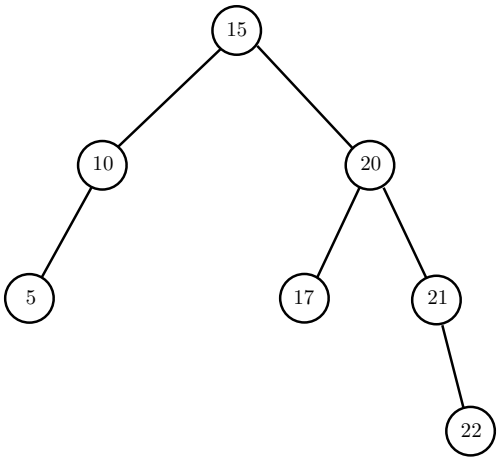


Abbildung 1: Der AVL-Baum

(2+2+2+2+2 Punkte)

Aufgabe 3 (Stacks und Permutationen): Bevor die Vorlesung demnächst behandelt, wie man sinnvollerweise sortiert, betrachten wir die Möglichkeit, vergleichbare[1] Elemente mittels eines Stapels zu ordnen. Betrachte dazu Algorithmus 1, der eine Sequenz $p_1, p_2, \ldots, p_n$ von vergleichbaren Elementen erhält und diese in (aufsteigend) sortierter Reihenfolge ausgeben soll. Zur Ausgabe eines Elements dient hier das Schlüsselwort **yield**.

```
 1: S ←Stack()
 2: for i ← 1, ..., n do
 3:     while not S.empty() and S.top() ≤ p_i do
 4:         yield S.pop()
 5:     end while
 6:     S.push(p_i)
 7: end for
 8: while not S.empty() do
 9:     yield S.pop()
10: end while
```

Algorithmus 1: Algorithmus zum Sortieren mit einem Stapel

a) Welche Laufzeit (in $\mathcal{O}$-Notation) hat Algorithmus 1 (mit Begründung!) für eine Sequenz der Länge n? Gib weiterhin eine kürzestmögliche Sequenz von natürlichen Zahlen an, die Algorithmus 1 nicht korrekt sortiert.

b) Zeige: Eine Sequenz $p_1, p_2, \ldots, p_n$ paarweise verschiedener Elemente wird genau dann von Algorithmus 1 korrekt sortiert, wenn für drei Elemente $p_i < p_j < p_k$ nie $j < k < i$ gilt.

[1] *Vergleichbar* bedeutet, dass für die Elemente eine Totalordnung vorliegt.

c) Angenommen, du hast zwei Stapel, S_1 und S_2, darfst aber weiterhin nur die Operationen `push(.)`, `pop()`, `empty()` und `top()` ausführen und hast keinen zusätzlichen Speicher. Formuliere einen Algorithmus, der mit Hilfe von S_1 und S_2 jede Folge von vergleichbaren Elementen korrekt sortiert. Welche Laufzeit (in $\mathcal{O}$-Notation) hat dein Algorithmus (mit Begründung!)?

Ähnlich zu der Frage nach sortierbaren Sequenzen ist die Frage, welche Permutationen[2] man mit einem Stapel aus der Folge $1, 2, \ldots, n$ generieren kann. Stellen wir `push(.)`-Operationen als 1 und `pop()`-Operationen als 0 dar, können wir eine Abfolge der Operationen als Binärstring der Länge $2n$ (n-mal `push(.)`, n-mal `pop()`) darstellen. Einige davon sind allerdings sinnlos, z.B. die Zeichenfolge 0011, da dort eine `pop()`-Operation auf einem leeren Stack ausgeführt wird und am Ende der Stack nicht leer ist. Eine Abfolge von Stack-Operationen, bei der nie `pop()` auf den leeren Stack aufgerufen wird und der Stack am Ende leer ist, nennen wir *zulässig*.

d) Gib eine einfache Charakterisierung der Binärstrings der Länge $2n$ an, die einer zulässigen Abfolge von Stack-Operationen entsprechen.

e) Zeige: Keine zwei unterschiedlichen zulässigen Abfolgen von Stack-Operationen führen zur selben Permutation.

(4+8+8+4+6 Punkte)

[2]Eine Permutation ist eine Anordnung der Zahlen $1, 2, \ldots, n$ in einer bestimmten Reihenfolge.