36. (7 Punkte) Wir wollen eine Variante von a-b-Bäumen konstruieren, bei der weniger Speicher verschwendet wird. Es sei $b = 100$. Wie groß kann man a wählen, wenn man
    - beim Entfernen bis zu *zwei* Geschwisterknoten zum Ausborgen in Betracht zieht, und
    - beim Einfügen einen Geschwisterknoten zu Hilfe nimmt, falls dieser den Überlauf aufnehmen kann?

37. (9 Punkte) *Rot-Schwarz-Bäume* sind binäre Bäume mit folgenden Eigenschaften:[1]
    (a) Jeder innere Knoten hat zwei Kinder.
    (b) Jeder Knoten ist entweder als *rot* oder als *schwarz* gekennzeichnet.
    (c) Die Wurzel und alle Blätter sind schwarz.
    (d) Die Kinder eines roten Knotens sind schwarz.
    (e) Ein schwarzer Knoten kann höchstens ein rotes Kind haben.
    (f) Alle Wege von der Wurzel zu den Blättern enthalten gleich viele schwarzen Knoten.

    Zeichnen Sie Rot-Schwarz-Bäume mit 5, 7 und 12 Blättern. Zeigen Sie, dass man aus jedem Rot-Schwarz-Baum einen 2-3-Baum machen kann, und umgekehrt.

38. (4 Punkte) Es sei A_h die kleinstmögliche Anzahl von Blättern eines AVL-Baumes mit Höhe h. Beweisen Sie, dass die Zahlen A_h die Fibonacci-Zahlen sind.

    Was können Sie daraus für die kleinstmögliche Anzahl $n(h)$ von *inneren Knoten* eines AVL-Baumes mit Höhe h schließen?

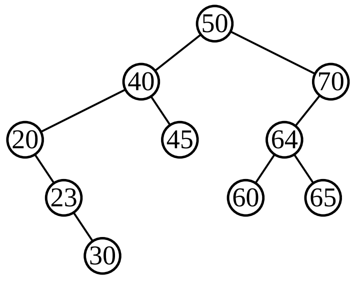
39. (0 Punkte) Wir wollen die Algorithmen zum Suchen, Einfügen und Löschen in einem binären Suchbaum so erweitern, dass man jederzeit auch das i-größte Element finden kann, und dass alle Operationen Zeit proportional zur Höhe des Baumes benötigen.

    Man könnte das lösen, indem man zu jedem Knoten den *Rang* speichert, das heißt, die Position, die dieser Schlüssel in der sortierten Reihenfolge einnimmt. Warum ist dies keine gute Idee?

40. (0 Punkte) Schreiben Sie ein Programm, das einen (nicht zu großen) binären Baum zweidimensional in übersichtlicher und schön lesbarer Form ausdruckt. Drei Vorschläge[2]:

```
      20   ----
         23   -
            30
   40   -------
      45
50  ----------
      64   -
         65
   70   ----
```

```
               50
           /----↥---------\
         40                 70
  /-------↥\            /---↥
20          45        64
 ↥\                  /-↥\
   23              60    65
   ↥-\
      30
```



[1]Es gibt verschiedene andere Versionen von Rot-Schwarz-Bäumen, bei denen zum Beispiel auch die inneren Knoten Werte enthalten (im Gegensatz zu den 2-3-Bäumen, wie sie in der Vorlesung besprochen wurden).
Eine Implementierung von Rot-Schwarz-Bäumen kann man in der Klasse `TreeMap` im Paket `java.util` finden, siehe `http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/TreeMap.java`.

[2]Das dritte Beispiel ist als PostScript-Datei erstellt worden, siehe
`http://www.inf.fu-berlin.de/~rote/Lere/2003-04-WS/Algorithmen+Programmierung3/baum.eps`.
Ihr Programm kann sich an dieser Datei als Beispiel orientieren.