Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Lösung - Übung 8

Tutoraufgabe 1 (AVL-Baum):

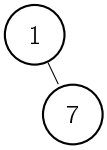
Führen Sie die folgenden Operationen beginnend mit einem anfangs leeren *AVL-Baum* aus und geben Sie die entstehenden Bäume nach jeder *Einfüge-* und *Löschoperation* sowie jeder *Rotation* an. Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird:

1. 1 einfügen
2. 7 einfügen
3. 6 einfügen
4. 5 einfügen
5. 4 einfügen
6. 3 einfügen
7. 2 einfügen
8. 8 einfügen
9. 10 einfügen
10. 9 einfügen
11. 11 einfügen
12. 1 löschen
13. 3 löschen
14. 2 löschen

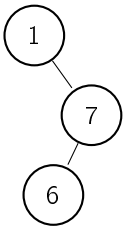
Lösung

füge 1 ein

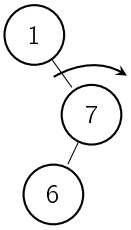
füge 7 ein

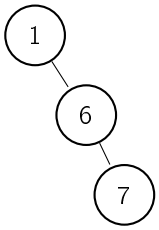


füge 6 ein

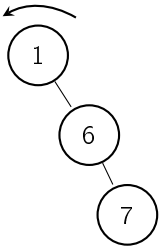


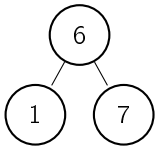
rotiere 7 nach rechts



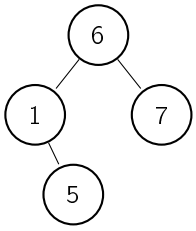


rotiere 1 nach links

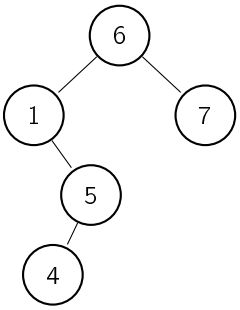




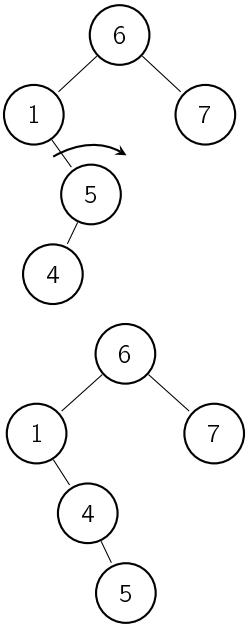
füge 5 ein



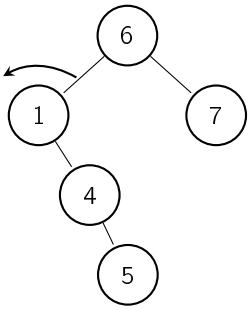
füge 4 ein

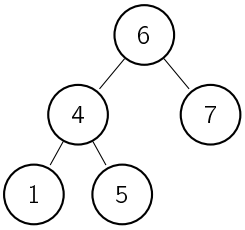


rotiere 5 nach rechts

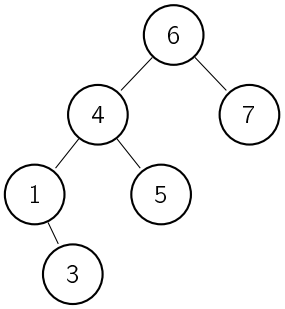


rotiere 1 nach links

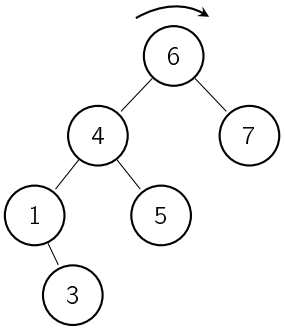


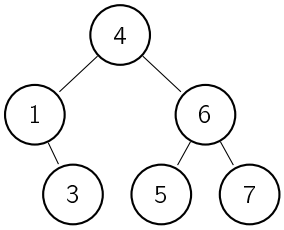


füge 3 ein

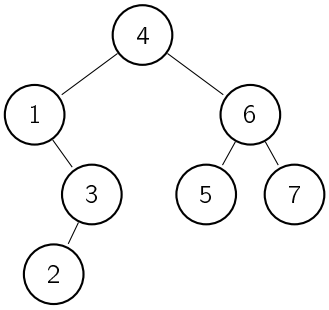


rotiere 6 nach rechts

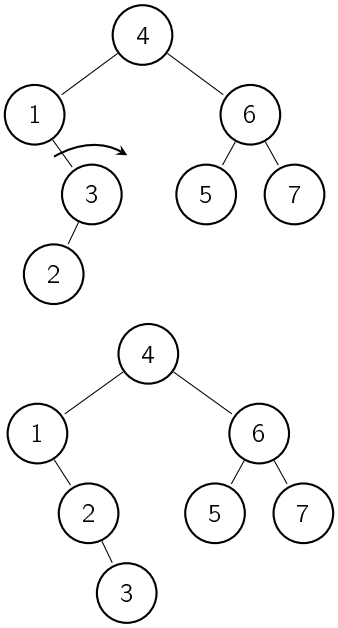




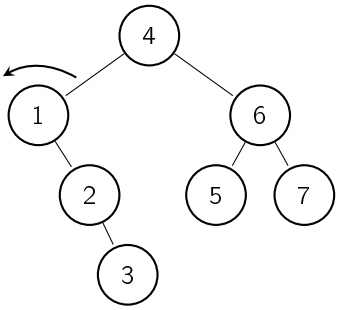
füge 2 ein

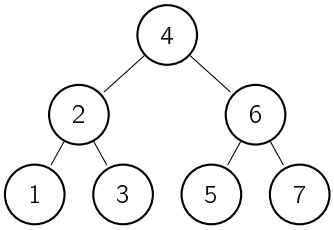


rotiere 3 nach rechts

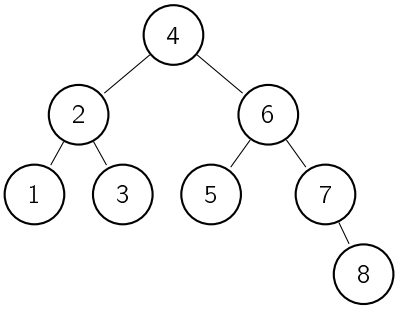


rotiere 1 nach links

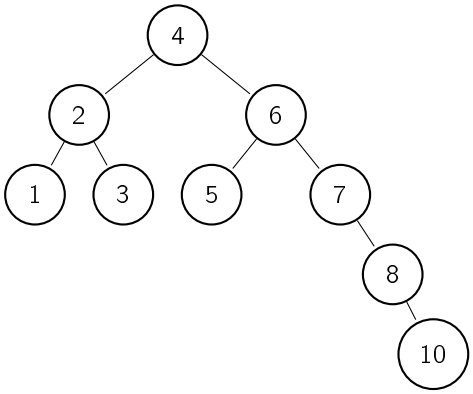




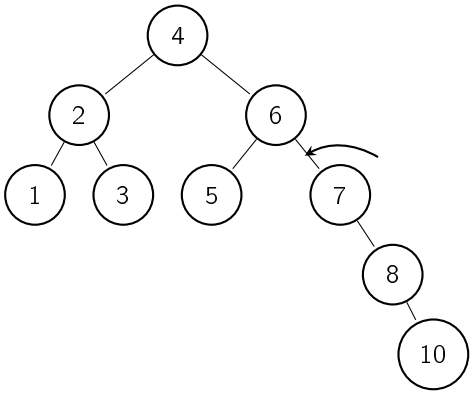
füge 8 ein

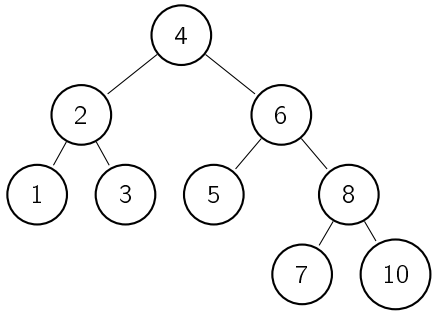


füge 10 ein

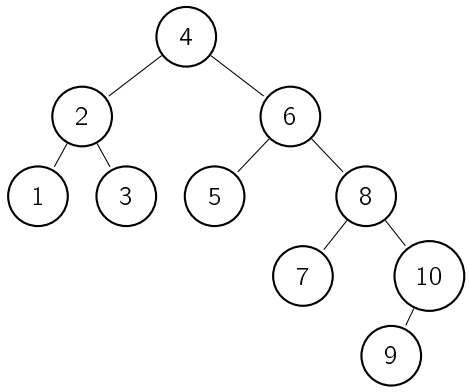


rotiere 7 nach links

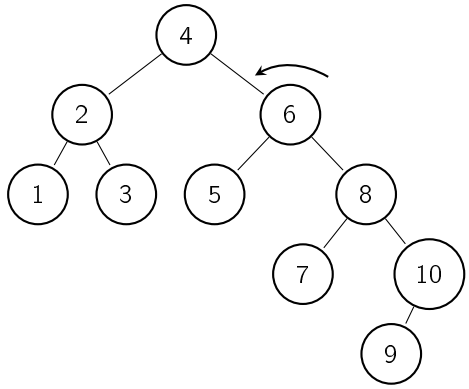


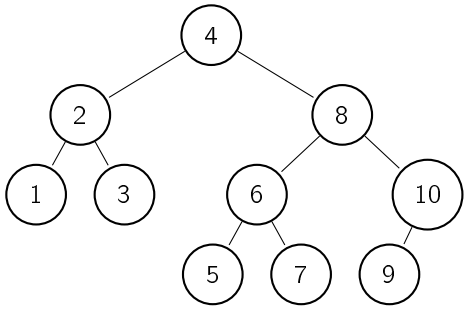


füge 9 ein

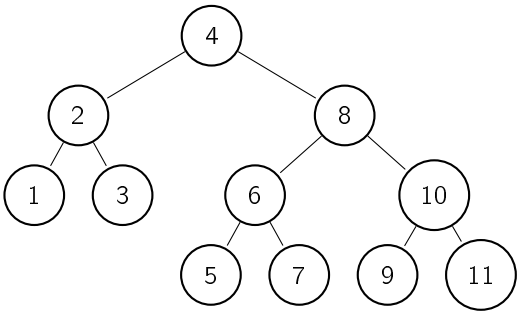


rotiere 6 nach links

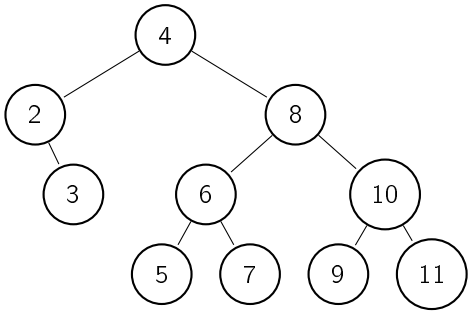




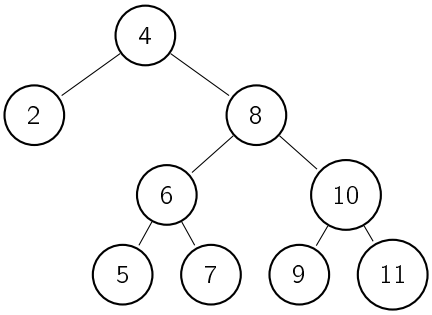
füge 11 ein



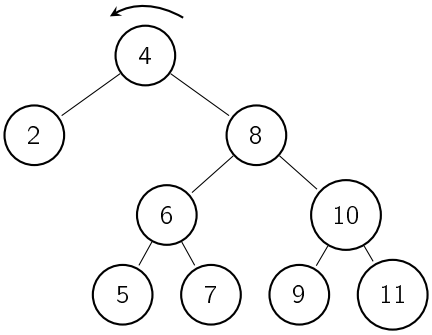
entferne 1

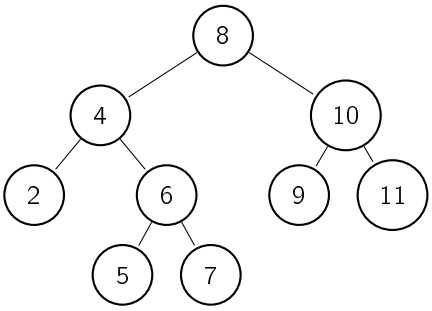


entferne 3

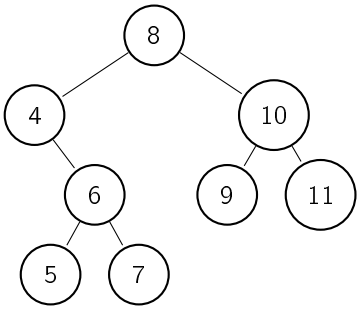


rotiere 4 nach links

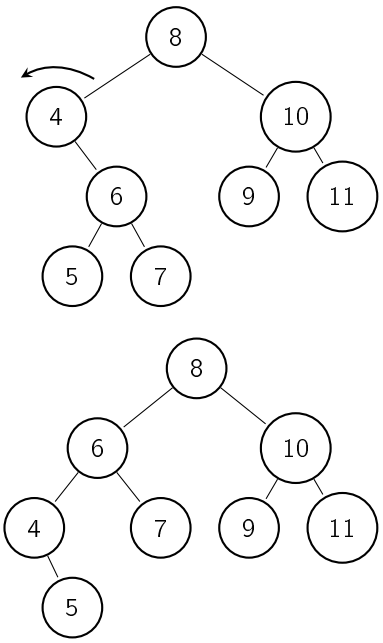




entferne 2



rotiere 4 nach links



Tutoraufgabe 2 (B-Baum):

Führen Sie folgenden Operationen beginnend mit einem anfangs leeren *B-Baum* mit Grad $t = 3$ aus und geben Sie die dabei jeweils entstehenden Bäume an:

1. 1 einfügen
2. 7 einfügen
3. 6 einfügen
4. 5 einfügen
5. 4 einfügen
6. 3 einfügen
7. 2 einfügen
8. 8 einfügen

9. 10 einfügen

10. 9 einfügen

11. 11 einfügen

12. 7 löschen

13. 8 löschen

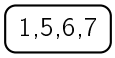
Lösung

Schritt 1: Füge 1 ein

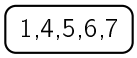
Schritt 2: Füge 7 ein

Schritt 3: Füge 6 ein

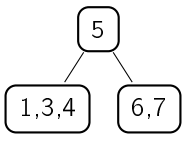
Schritt 4: Füge 5 ein



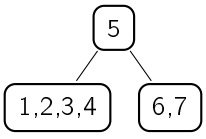
Schritt 5: Füge 4 ein



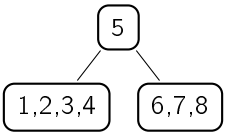
Schritt 6: Füge 3 ein



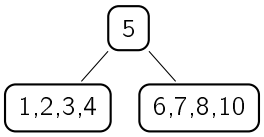
Schritt 7: Füge 2 ein



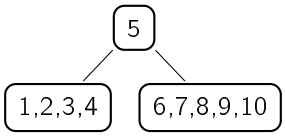
Schritt 8: Füge 8 ein



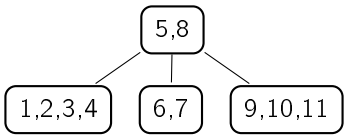
Schritt 9: Füge 10 ein



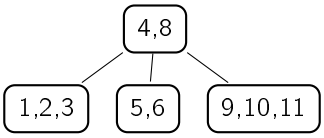
Schritt 10: Füge 9 ein



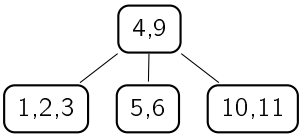
Schritt 11: Füge 11 ein



Schritt 12: Lösche 7



Schritt 13: Lösche 8



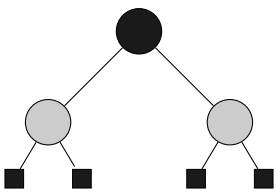
Tutoraufgabe 3 (Rot-Schwarz Bäume – Theorie):

Die Anzahl der schwarzen Knoten auf einem Pfad von der Wurzel eines Rot-Schwarz Baums bis zu einem Blatt (exclusive der Wurzel), ist für einen gegebenen Rot-Schwarz Baum immer dieselbe, unabhängig vom gewählten Pfad. Die Schwarzhöhe eines Rot-Schwarz Baums ist definiert als eben diese Anzahl.

Beweisen Sie (per Induktion) die folgende Aussage:
Ein Rot-Schwarz-Baum der Schwarzhöhe h hat maximal $rot(h) = \frac{2}{3} \cdot 4^h - \frac{2}{3}$ rote Knoten.

Lösung

Beweis durch vollständige Induktion über die Schwarzhöhe h.
Induktionsverankerung für $h = 1$:
Der Rot-Schwarz Baum mit Schwarzhöhe $h = 1$ mit maximaler Anzahl an roten Knoten ist der folgende:

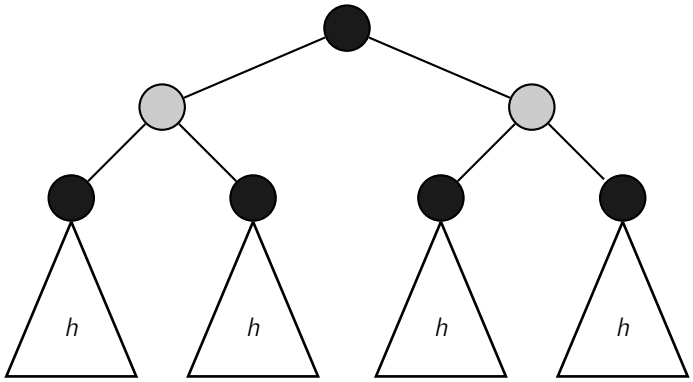


Dieser Baum besitzt zwei rote Knoten. Ebenfalls ergibt sich $rot(1) = \frac{2}{3} \cdot 4^1 - \frac{2}{3} = \frac{8}{3} - \frac{2}{3} = 2$.

Induktionsverankerung für $h = 0$ (alternativ):
Es gibt nur einen Rot-Schwarz Baum mit Schwarzhöhe $h = 0$. Dieser besteht nur aus der Wurzel. Da diese Schwarz ist enthält er keine roten Knoten. Aus der zu beweisenden Formel ergibt sich ebenfalls $rot(0) = \frac{2}{3} \cdot 4^0 - \frac{2}{3} = \frac{2}{3} - \frac{2}{3} = 0$.

Induktionsvoraussetzung:
In einem Rot-Schwarz Baum der Schwarzhöhe h gibt es $rot(h) = \frac{2}{3} \cdot 4^h - \frac{2}{3}$ viele rote Knoten.

Induktionsschritt von h nach $h+1$:
Um einen Baum mit maximaler Anzahl an roten Knoten zu erhalten, müssen die beiden Kinder des Wurzelknotens rot sein. Wäre dies nicht der Fall, so könnten wir einen roten Knoten dazwischen packen, ohne die Schwarzhöhe zu verändern. Um einen Baum der Schwarzhöhe $h+1$ mit maximaler Anzahl an Rotknoten zu erhalten, bilden wir den folgenden Baum:

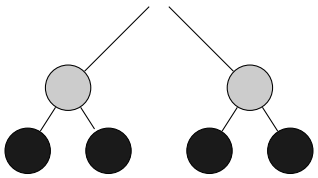


Die Anzahl der roten Knoten in diesem Baum berechnet sich nun wie folgt:

$$\begin{aligned} rot(h+1) &= \overbrace{4 \cdot rot(h)}^{\text{rote Knoten in den Teilbäumen}} + \overbrace{2}^{\text{Knoten auf der 1. Ebene}} \\ &\overset{I.V.}{=} 4 \cdot (\tfrac{2}{3} \cdot 4^h - \tfrac{2}{3}) + 2 \\ &= \tfrac{2}{3} \cdot 4^{h+1} - \tfrac{8}{3} + \tfrac{6}{3} \\ &= \tfrac{2}{3} \cdot 4^{h+1} - \tfrac{2}{3} \end{aligned}$$

Somit ist die Aussage bewiesen.

Konstruktiver Beweis
Ein konstruktiver Beweis funktioniert wie folgt:
Um einen Baum mit maximaler Anzahl an roten Knoten zu erhalten, müssen die beiden Kinder jedes schwarzen Knotens rot sein. Wäre dies nicht der Fall, so könnten wir einen roten Knoten dazwischen packen, ohne die Schwarzhöhe zu verändern. Wir können den zu untersuchenden Rot-Schwarz Baum also so konstruieren, dass wir mit der Wurzel beginnen und jede weitere Ebene des Baums daraus entsteht, dass an jedes Blatt folgende Struktur angehängt wird:



Bis auf die Wurzel besteht der Baum also nur aus obigen Strukturen. In der Struktur ist die Anzahl der schwarzen Knoten doppelt so groß wie die Anzahl der roten Knoten. Wird also die Anzahl der schwarzen Knoten im gesamten Baum um eins reduziert (die Wurzel), so erhalten wir eine Zahl die doppelt so groß ist wie die Anzahl der roten Knoten im gesamten Baum ($rot(h)$).
Berechnen wir nun die Anzahl der schwarzen Knoten im gesamten Baum mit Schwarzhöhe h. Wir erkennen, dass dies Äquivalent ist zu der Frage wie viele Knoten in einem vollständigen Baum der Höhe h mit Grad 4 sind. Der Grad 4 ergibt sich daraus, dass wir obige Struktur an jeden schwarzen Knoten anhängen und wir somit an jeden schwarzen Knoten vier weitere schwarze Knoten anhängen. Auf der obersten Ebene haben wir $4^0 = 1$ Knoten, darunter $4^1 = 4$ Knoten, darunter $4^2 = 16$ Knoten bis wir auf der letzten Ebene mit 4^h Knoten schließen. Summieren wir dies auf, so ergibt sich mithile der geometrischen Reihe folgende Knotenanzahl für den gesamten Baum:

$$\sum_{i=0}^{h} 4^i = \frac{4^{h+1} - 1}{4 - 1}$$

$\frac{4^{h+1}-1}{3}$ ist also die Anzahl der schwarzen Knoten im untersuchten Rot-Schwarz Baum. Daraus ergibt sich nun sofort die Anzahl der roten Knoten im untersuchten Baum indem wir, wie oben beschrieben, zunächst eins abziehen und anschließend die Zahl halbieren:

$$rot(h) = \frac{\frac{4^{h+1}-1}{3} - 1}{2} = \frac{\frac{4 \cdot 4^h}{3} - \frac{1}{3} - \frac{3}{3}}{2} = \frac{4}{3 \cdot 2} \cdot 4^h - \frac{4}{3 \cdot 2} = \frac{2}{3} \cdot 4^h - \frac{2}{3}$$

Aufgabe 4 (B-Baum): 10 Punkte

Führen Sie folgenden Operationen beginnend mit einem anfangs leeren *B-Baum* mit Grad $t = 3$ aus und geben Sie die dabei jeweils entstehenden Bäume an:

1. 9 einfügen
2. 6 einfügen
3. 8 einfügen
4. 7 einfügen
5. 2 einfügen
6. 3 einfügen
7. 4 einfügen

8. 5 einfügen
9. 1 einfügen
10. 12 einfügen
11. 8 löschen
12. 5 löschen
13. 7 löschen
14. 3 löschen
15. 9 löschen

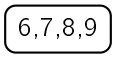
Lösung

Schritt 1: Füge 9 ein

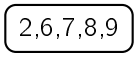
Schritt 2: Füge 6 ein

Schritt 3: Füge 8 ein

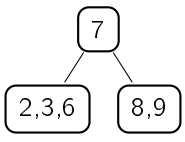
Schritt 4: Füge 7 ein



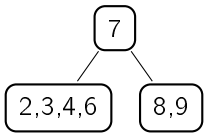
Schritt 5: Füge 2 ein



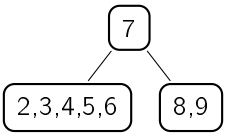
Schritt 6: Füge 3 ein



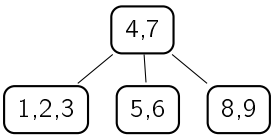
Schritt 7: Füge 4 ein



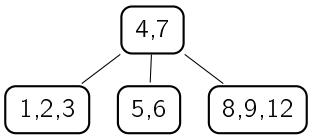
Schritt 8: Füge 5 ein



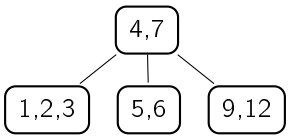
Schritt 9: Füge 1 ein



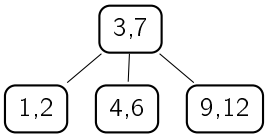
Schritt 10: Füge 12 ein



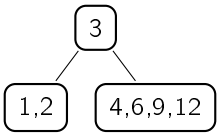
Schritt 11: Lösche 8



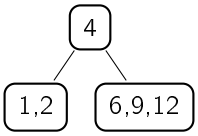
Schritt 12: Lösche 5



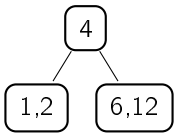
Schritt 13: Lösche 7



Schritt 14: Lösche 3



Schritt 15: Lösche 9



Aufgabe 5 (Max 3 Skip-Liste): 3 + 5 + 2 = 10 Punkte

Sei eine Skip-Liste mit unterschiedlichen Elementen $k_1 \dots k_n$ gegeben. Die Anzahl der Vorwärtskanten des Elements k_i ist gegeben durch $level(k_i)$. Wir definieren nun eine Max 3 Skip-Liste als eine solche Skip-Liste, in der maximal 3 Elemente mit einer Höhe ohne Unterbrechung eines größeren Level vorkommen darf und dass kein level unnötig ist. Mit unnötig meinen wir, dass es zwischen zwei Elementen mit einer Vorwärtskante (oder dem Anfang, bzw das Ende) es mindestens ein Element für jedes kleinere Level gibt. Formal ist eine Skip-Liste genau dann eine Max 3 Skip-Liste

1. wenn für alle $i < i' < i''$ mit $level(k_i) = level(k_{i'}) = level(k_{i''})$ gilt,
   a) dass ein j mit $i < j < i''$ existiert sodass $level(k_j) > level(k_i), level(k_{i'}), level(k_{i''})$, oder
   b) dass kein j mit $i < j < i''$ und $j \neq i'$ existiert sodass $level(k_j) = level(k_i), level(k_{i'}), level(k_{i''})$; und
2. wenn für jedes i alles vier gilt:
   a) dass es für jedes $l < level(k_i)$ ein Element k_j mit $level(k_j) = l$ und $j < i$ existiert,
   b) dass es für jedes $l < level(k_i)$ ein Element k_j mit $level(k_j) = l$ und $j > i$ existiert,
   c) dass für jedes i' mit $i < i'$ und $level(k_i) \leq level(k_{i'})$ und jedes $l < level(k_i)$ ein Element k_j mit $level(k_j) = l$ und $i < j < i'$ existiert, und
   d) dass für jedes i' mit $i > i'$ und $level(k_i) \leq level(k_{i'})$ und jedes $l < level(k_i)$ ein Element k_j mit $level(k_j) = l$ und $i > j > i'$ existiert.

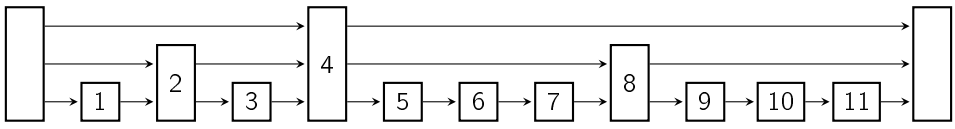
a) Geben Sie ein Beispiel einer Max 3 Skip-Liste mit 11 Elementen an, in der genau ein Element Level 3 hat.

b) Geben Sie an, wie eine Max 3 Skip-Liste mit maximalem Level l in ein B-Baum mit minimalen Grad 2 und Höhe $l - 1$ transformiert werden kann. Begründen Sie auch die Korrektheit ihrer Transformation, das heißt begründen Sie,
   i. dass die Elemente der Skip-Liste die Elemente des B-Baums sind,
   ii. dass der B-Baum minimalen Grad 2 hat, und
   iii. dass der B-Baum Höhe $l - 1$ hat.

c) Erläutern Sie, wie Sie aus ihrer Transformation von Max 3 Skip-Listen auf B-Bäume Methoden zum deterministischen Einfügen und Löschen in einer Max 3 Skip-Liste herleiten können. Es ist nicht nötig, die Methoden im Detail zu erläutern.

Lösung

a) Ein Beispiel für eine Max 3 Skip-Liste mit genau 11 Elementen und genau einem Element mit Level 3 ist folgendes:



b) Wir transformieren eine Max 3 Skip-Liste mit maximalem Level l in einen B-Baum mit Grad 2 und Höhe l indem wir alle Elemente k_i mit level l_i einem Knoten auf Tiefe $l - l_i$ hinzufügen.

Genauer gehen wir wie folgt vor:

- Sei l das maximale Level der Skip-Liste
- Wir fügen alle Elemente k mit $level(k) = l$ in einen B-Baum Knoten.
- Seien diese Elemente k_h, k_i, k_j (falls es weniger sind, ist der Algorithmus analog)
- Wir teilen die Skip-Liste in 4 neue Skip-Listen:
  a) Alle Elemente k mit $k < k_h$
  b) Alle Elemente k mit $k_h < k < k_i$
  c) Alle Elemente k mit $k_i < k < k_j$
  d) Alle Elemente k mit $k_j < k$
- Transformiere die 4 Skip-Listen rekursiv in 4 B-Bäume
- Hänge die 4 B-Bäume jeweils an die Wurzel mit k_h, k_i und k_j an.

Zwecks Korrektheit haben wir drei Eigenschaften zu prüfen:

i. Alle Elemente in der Skip-Liste kommen auch im B-Baum genau einmal vor, da Sie entweder k_h, k_i oder k_j sind oder in eine der 4 Skip-Liste danach auftauchen. Ist das Element in der Wurzel, haben weitere Rekursionsaufrufe kein Zugriff mehr auf diese Elemente. Tauchen sie in eine der 4 Skip-Listen auf, so garantiert der Rekursive Aufrufe dass sie genau 1 mal in einen B-Baum hinzugefügt werden.

ii. Der B-Baum hat minimalen Grad 2.

- In jedem Rekursionsaufruf haben wir maximal 3 Elemente mit maximalem Level. Dies wird durch die ersten Eigenschaft der Max 3 Skip-Liste garantiert: Gäbe es mehr als 3 Elemente mit maximalen Level würde diese nicht gelten. Weiterhin müssen auch die Teillisten Max 3 Skip-Listen sein, da k_h, k_i und k_j Seperatoren sind, aber für das Level darunter nur maximal 3 Elemente auf deren Teil Maximalem Level sein können.
- In jedem Rekursionsaufruf haben wir mindestens 1 Element mit maximalem Level - sonst wäre das maximale Level nicht das maximale Level.

iii. Der B-Baum hat Höhe $l - 1$. Da nach der zweiten Bedingung für Max 3 Skip-Listen alle Teil-Skip-Listen alle niedrigeren Level besetzt sind und damit das höchste Level um eins kleiner wird und die Höhe des Baums um eines erhört wird. Die Höhe des Baumes ist jedoch eines weniger als es Ebenen hat, damit auch -1.

c) Tatsächlich können wir die Transformation aus der vorherigen Aufgabe auch leicht wieder invertieren. Dadurch müssen wir uns in jedem Knoten lediglich merken, welches Level es hatte und können anschließend die Skip-Liste wieder rekonstruieren indem wir wieder rekursiv durch den Baum gehen und Elemente mit entsprechendem Level erstellen. Dadurch können wir Einfüge- und Löschoperationen aber auch auf dem B-Baum anwenden. Müssen wir einen Knoten aufteilen, so erhöhen wir die Level des neuen Knoten und allen Elternknoten entsprechend. Fügen wir Knoten wieder zusammen, müssen wir die Level der Elternknoten wieder entsprechend senken.

Hinweis:

- Tatsächlich ist es auch möglich, diese Methoden direkt auf der Skip-Liste anzuwenden, sodass man sich die Transformation in den B-Baum und zurück in die Skip-Liste spart[a].

Aufgabe 6 (Programmierung in Python - AVL-Bäume): 2 + 6 + 6 + 6 = 20 Punkte

Bearbeiten Sie die Python Programmieraufgaben. In dieser praktischen Aufgabe werden Sie sich mit AVL-Bäumen auseindersetzen. Diese Aufgabe dient dazu einige Konzepte der Vorlesung zu wiederholen und zu vertiefen.
Zum Bearbeiten der Programmieraufgabe können Sie einfach den Anweisungen des Notebooks *blatt08-python.ipynb* folgen. Das Notebook steht in der .zip-Datei zum Übungsblatt im Lernraum zur Vergügung.
Ihre Implementierung soll immer nach dem # *YOUR CODE HERE* Statement kommen. Ändern Sie keine weiteren

Zellen.
Laden Sie spätestens bis zur Deadline dieses Übungsblatts auch Ihre Lösung der Programmieraufgabe im Lernraum hoch. Die Lösung des Übungsblatts und die Lösung der Programmieraufgabe muss im Lernraum an derselben Stelle hochgeladen werden. Die Lösung des Übungsblatts muss dazu als .pdf-Datei hochgeladen werden. Die Lösung der Programmieraufgabe muss als .ipynb-Datei hochgeladen werden.

Übersicht der zu bearbeitenden Aufgaben:

a) AVL-Bäume

- get_balance()
- rotate_left()
- rotate_right()
- balance()

Lösung

Die Lösung der Programmieraufgaben finden Sie im Lernraum. Die Datei trägt den Namen *blatt08-python-solution.ipynb*.