Algorithmische Anwendungen WS 2006/2007

Praktikum 3:

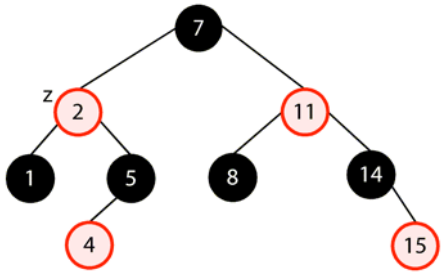
- **Aufgabe 1:** Einfügen eines Knotens ist einen Rot-Schwarz-Baum
- **Aufgabe 2**: Erklärungen für die Beobachtungen auf Folie 2.1/67
- **Aufgabe 3**: Zeige wie RB-DELETE-FIXUP die Eigenschaften 2 und 4 „repariert“
- **Aufgabe 4**: Beispiele zum Löschen von Knoten
- **Aufgabe 5**: Einfügen und Löschen mehrerer Knoten
- **Aufgabe 6:** Löschen von Knoten in Treaps

Literatur: Kapitel 2.1 des Skripts „Rot-Schwarz-Bäume“
Kapitel 2.2 des Skripts „Treaps“
Cormen. Kapitel 13, Red-Black-Trees

Das sollten Sie für Praktikum 3 lesen und wissen!

Aufgabe 1 (Einfügen eines Knotens)

Gegeben ist folgender Rot-Schwarz-Baum:



a) Fügen Sie den Schlüssel 9 ein und prüfen Sie die Rot-Schwarz-Eigenschaften. Verfolgen und erklären Sie Schritt für Schritt den Ablauf der Pseudocodes RB-INSERT und RB-INSERT-FIXUP. Prüfen Sie alle Bedingungen für den Aufruf der Funktionen und für die Ausführung der Fälle. Skizzieren Sie den Ablauf auf Papier anhand des Beispiels.

b) Fügen Sie in den unter a) entstandenen Baum den Schlüssel 17 ein. Verfolgen Sie ebenfalls den Ablauf der Pseudocodes für die Funktionen RB-INSERT und RB-INSERT-FIXUP. Zeichnen Sie die RS-Bäume mit allen Zwischenschritten.

Hinweis: Beachten Sie, dass in der then-Klause von RB-INSERT-FIXUP (Zeilen 3-14) *left* und *right* vertauscht werden müssen, falls p[z] rechter (und nicht linker) Sohn seines Vaters p[p[z]] ist. Siehe auch Zeile 15.

Aufgabe 2 (Erklärungen für die Beobachtungen auf Folie 2.1/67)

Arbeiten Sie das Skript zu Kapitel 2.1 durch (Folien 53-74).

a) Analysieren Sie die Pseudocodes der beiden Funktionen RB-DELETE(T, z) und RB-DELETE-FIXUP(T, x). Erklären Sie Beobachtung 1 auf Folie 2.1/67: „Innerhalb der while-Scheife von RB-DELETE-FIXUP ist x immer ein *schwarz-schwarz*-Knoten, und x ist nicht die Wurzel.“ Erklären Sie genau, warum das so ist.

b) Erklären Sie Beobachtung 2 auf Folie 2.1/67: „Da x *schwarz-schwarz* ist, kann w nicht der Sentinel nil[T] sein.“

Aufgabe 3 (RB-DELETE-FIXUP „repariert“ Eigenschaften 2 und 4)

a) Erklären Sie, warum nach Ausführung der while-Schleife in RB-DELETE-FIXUP die Wurzel des Baumes schwarz sein muss, d.h. Eigenschaft 2 ist erfüllt.

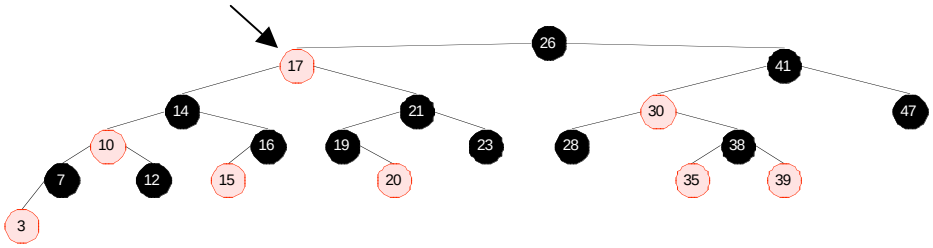
b) Erklären Sie, warum nach der Ausführung der while-Schleife in RB-DELETE-FIXUP keine zwei roten Knoten aufeinander folgen können, d.h. Eigenschaft 4 ist erfüllt.

Hinweis:

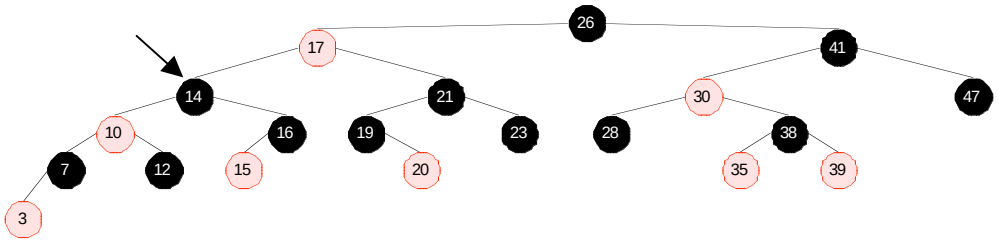
Lösen Sie die Aufgaben a) und b) anhand von Beispielen. Zeichnen Sie für die 4 Fälle in RB-DELETE-FIXUP die Situationen vor und nach der Anwendung der jeweiligen Fälle.

Aufgabe 4 (Löschen eines Knotens)

a) Löschen Sie aus dem folgenden Rot-Schwarz-Baum den Knoten 17 und führen reorganisieren Sie den Baum schrittweise. Erklären Sie genau die einzelnen Schritte. Verfolgen und erklären Sie die Fälle der Funktion RB-DELETE-FIXUP.



b) Löschen Sie aus dem folgenden Rot-Schwarz-Baum den Knoten 14 und reorganisieren Sie den Baum.



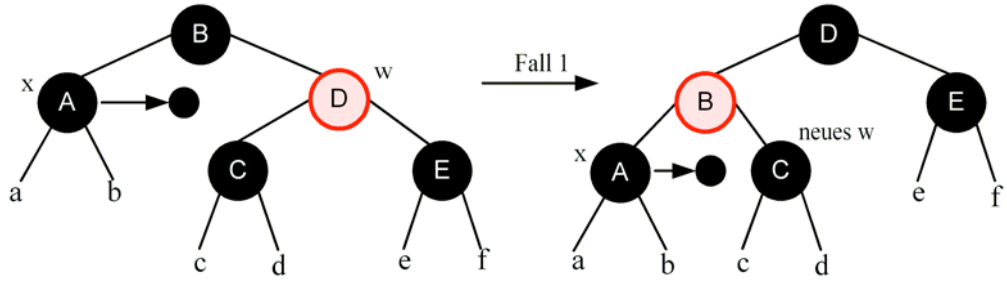
Aufgabe 5 (Einfügen und Löschen mehrer Knoten)

a) Zeichnen und erklären Sie die Rot-Schwarz-Bäume, die durch Einfügen der Schlüssel 41, 38, 31, 12, 19, 8 – in genau dieser Reihenfolge – in den anfangs leeren Baum entstehen.

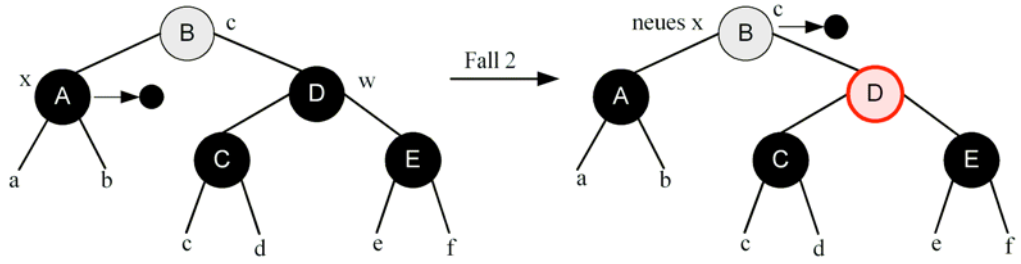
b) Zeichnen und erklären Sie die Rot-Schwarz-Bäume, die durch Löschen der Schlüssel 8, 12, 19, 31, 38, 41 – in genau dieser Reihenfolge – aus dem unter a) konstruierten Baum entstehen.

Die vier Fallunterscheidungen in RB-DELETE-FIXUP(T, x):

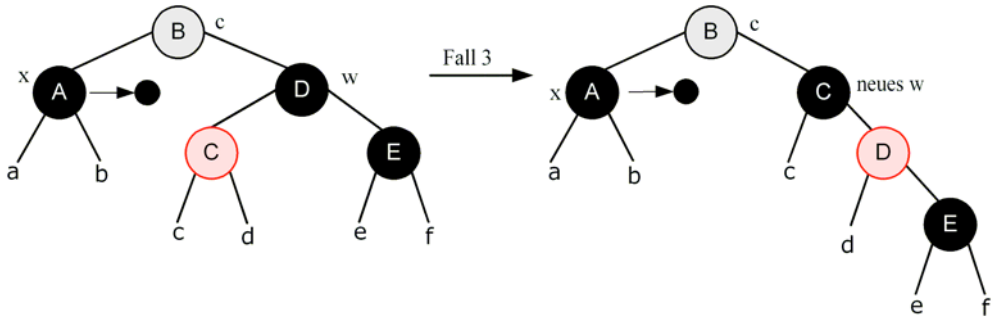
Fall 1: x‘s Bruder w ist rot



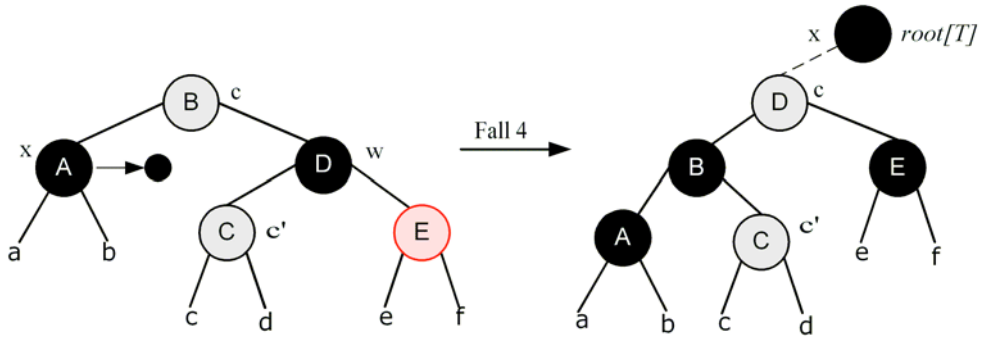
Fall 2: x‘s Bruder w ist schwarz und beide Kinder von w sind schwarz



Fall 3: x‘s Bruder w ist schwarz, w‘s linkes Kind ist rot und w‘s rechtes Kind ist schwarz



Fall 4: x‘s Bruder w ist schwarz und das rechte Kind von w ist rot



Graue Knoten (z.B. in Fall 4 links B u. C) können ROT oder SCHWARZ sein!

Aufgabe 6 (Löschen in Treaps)

In der Vorlesung wurde die Operation **`insert(x`***)* für Treaps besprochen, sie besteht aus einer Suchphase und einer oder mehreren Rotationsphasen. **`insert(x)`** kann im Wesentlichen in 3 Schritten realisiert werden. Dabei ist **`k=key(x)`** der Schlüssel von x und **`prio(x)`** die dazu zufällig ermittelte Priorität.

(1) Suche die Position für das Element x ohne die Prioritäten zu beachten, d.h. mit virtueller Priorität ∞.

(2) Erzeuge an dieser Position einen neuen Knoten für x. Jetzt ist x eventuell ein „schlechter“ Knoten, da die Priorität von x kleiner als die Priorität seines Vaters sein kann, d.h. **`prio(x) < prio(p(x))`**.

(3) Wiederherstellen der Priority-Queue-Eigenschaft.

```
while prio(p(x)) > prio(x) do
   if x rechter Sohn
     then Linksrotation(p(x))
     else Rechtsrotation(p(x))
```

Es gilt folgendes **Lemma**:

1. Links- und Rechtsrotationen lassen die Suchbaumstruktur intakt und respektieren alle Prioritätsregeln außer die für x und dessen Vater p(x).
2. Beim Abbrechen der Schleife in (3) liegt wieder ein Treap vor.

a) Realisieren Sie, ähnlich wie bei **`insert(x)`** die Operation **`delete(k)`** als Pseudocode, wobei **`k=key(x)`**.

b) Stellen ein entsprechendes Lemma für **`delete(k)`** auf.
Welcher Zusammenhang besteht zwischen den Operationen von **`insert(x)`** und denen von **`delete(k)`**?

c) Führen Sie **`delete(k)`** für das Element **`k=key(x)=B`** des folgenden Treaps aus. Zeichnen Sie dazu die einzelnen Schritte und erklären Sie diese.