*Dieses Praktikum beschäftigt sich mit den Inhalten aus der Vorlesung **ADS_K6_BalancierteBaeume**.*

3 Praktikum: Rot-Schwarz-Baum

Sie sind nach wie vor als Systemarchitekt bei „Data Fuse Inc.“ und konnten Ihre Kollegen bereits mit den Programm zur Datenhaltung und -verarbeitung überzeugen. Es hat sich aber herausgestellt, dass ein Binärbaum zur Datenhaltung hier nicht optimal ist. Bei großen Datenmengen kann es zu entarteten Bäumen kommen. Um z.B. die Suche zu optimieren, sollen Sie dies durch die Organisation mittels eines ausgeglichenen Baumes ersetzen. Hierzu wählen Sie einen Rot-Schwarz-Baum mit der Top-Down Einfügemethode.

3.1 Aufgabenstellung

Übernehmen Sie Baum- und Knotenklasse aus dem 1. Praktikum und erweitern Sie die Klassen wie in den UML Diagrammen angegeben (neue bzw. entfernte Attribute/Methoden sind in rot markiert).

Beachten Sie ALLE Lösungshinweise und setzen Sie sich zuerst mit der Funktionsweise eines Rot-Schwarz-Baumes auseinander.

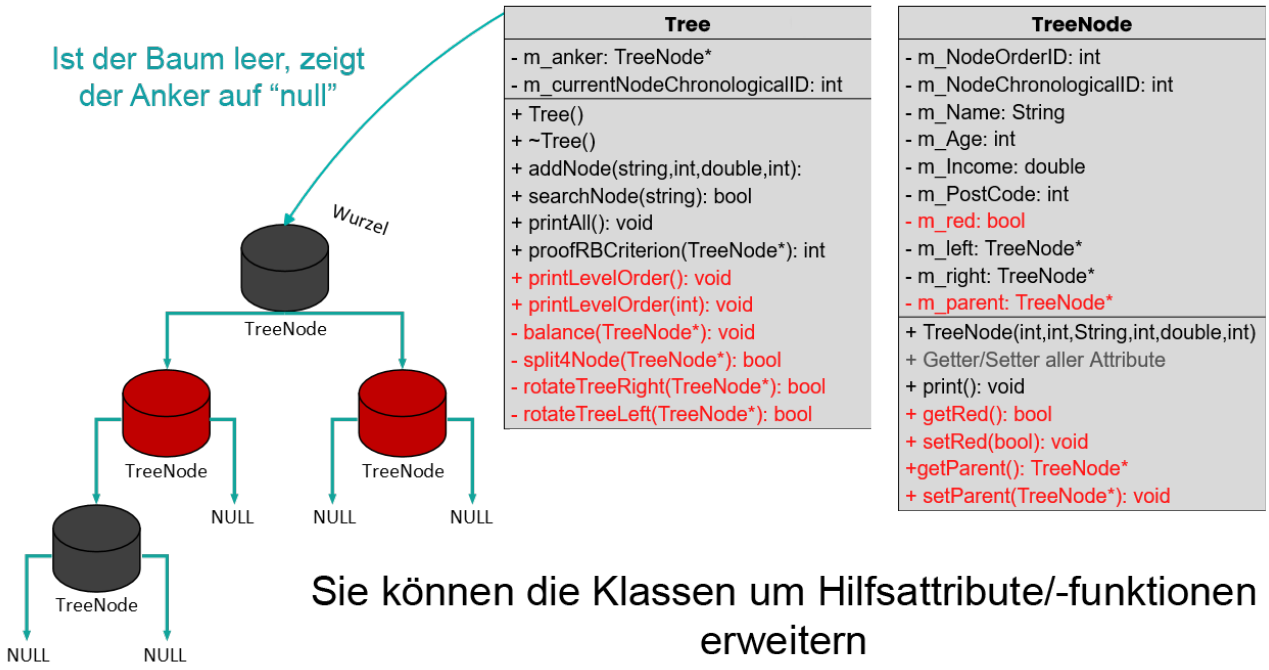

Abbildung 1: Aufbau der Baumstruktur und Erweiterung der Klassen.

In der Vorlesung wurde zunächst der 2-3-4-Baum vorgestellt. Das ist ein geordneter Suchbaum, bei dem die Knoten einen, zwei oder drei Schlüssel aufnehmen und entsprechend zwei, drei oder vier Nachfolger haben können. Die Implementierung eines 2-3-4-Baumes ist mit einem Rot-Schwarz-Baum möglich. So lässt sich ein gültiger Rot-Schwarz-Baum jederzeit in einen 2-3-4-Baum überführen und umgekehrt. Ein Rot-Schwarz-Baum ist ein binärer Suchbaum mit folgenden zusätzlichen Eigenschaften:

- Ein knoten hat die Farbe rot oder schwarz. Ein roter Knoten gehört zu seinem schwarzen Elternknoten und befindet sich im Sinne der 2-3-4-Baumes auf dem gleichen Niveau.
- Ein roter Knoten hat immer einen schwarzen oder keinen Nachfolger. Ein Baum mit 2 aufeinander folgenden roten Knoten ist kein gültiger Rot-Schwarz-Baum
- Die Wurzel ist immer schwarz.
- Ein neuer Knoten wird immer als roter Knoten und als Blattknoten eingefügt.
- Alle Pfade von der Wurzel bis zum Blatt haben die gleiche Anzahl schwarzer Knoten. Dies wird als Schwarzausgeglichenheit bezeichnet.

Die folgenden Abbildungen 2 bis 7 zeigen mögliche Transformationen von 3-er und 4-er Knoten aus einem 2-3-4-Baum in Rot-Schwarz-Bäume, sowie einen Einfüge-Fall in einen Rot-Schwarz-Baum.

Eine Hauptaufgabe in diesem Praktikum ist das Einfügen eines neuen Knotens in einen Rot-Schwarz-Baum, sowie die niveauweise Ausgabe der Knoten des Rot-Schwarz-Baumes im Sinne des 2-3-4-Baumes.

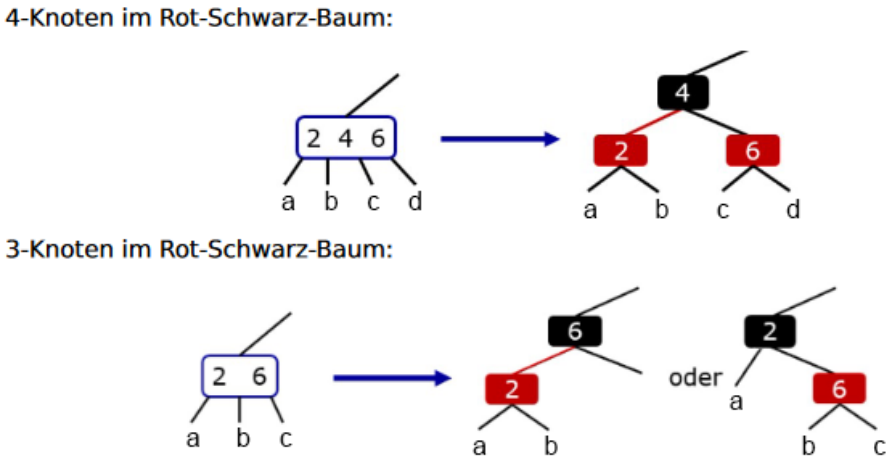

Abbildung 2: 2-3-4-Baum als Rot-Schwarz-Baum. Abbildungen von 2-3-4-Knoten.

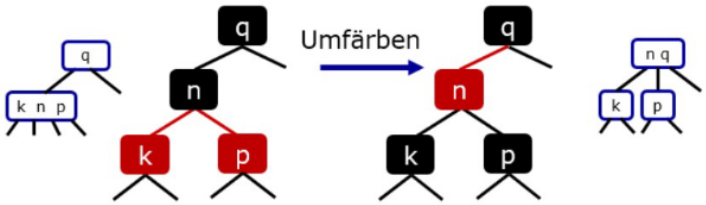

Abbildung 3: Umwandlung des Elternknoten (q) (2-Knoten) in einen 3-Knoten (nq) entspricht dem Umfärben der Knoten. (k,n,p)

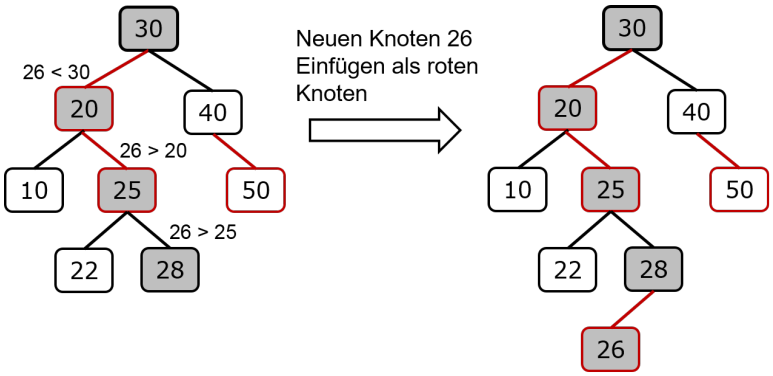

Abbildung 4: Einfügen des neuen Knoten 26 als roten Blattknoten. Der Suchpfad bis zum Knoten 28 bzw. 26 ist grau markiert. Alle 4er Knoten entlang des Suchpfades werden umgefärbt. Der neue Knoten 26 wird als roter Blattknoten entsprechend der Ordnung eingefügt.

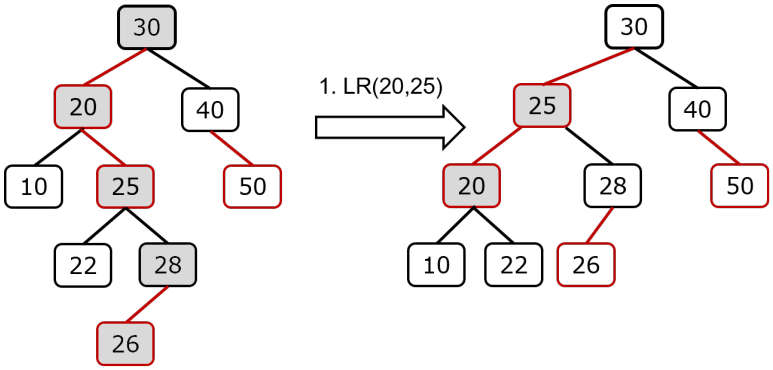

Abbildung 5: Bottom-Up-Verfolgung des Suchpfades vom Blatt-Knoten 26 hoch bis zur Wurzel: Wenn dabei 2 aufeinander folgende rote Knoten im Suchpfad detektiert werden, müssen Sie eine oder zwei Rotationen durchführen. Da diese hier bei den Knoten 30, 20, 25 im Links-Rechts-Weg(Knick) erfolgen, wird eine Doppelrotation durchgeführt. Hier: Die erste Rotation ist eine Links-Rotation zwischen den Knoten 20 und 25.

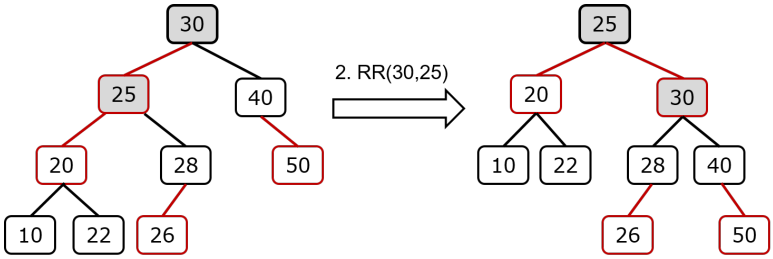

Abbildung 6: Durchführung der zweiten Rotation. Hier: Rechts-Rotation zwischen den Knoten 30 und 25.

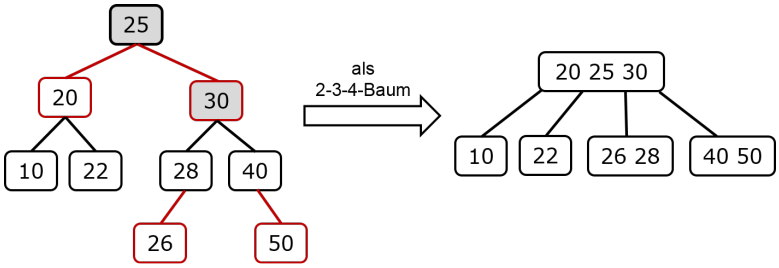

Abbildung 7: Ergebnis nach dem Einfügen des neuen Knoten 26 als Rot-Schwarz-Baum und als 2-3-4-Baum.

Die zu implementierenden Funktionen übernehmen dabei folgende Aufgaben:

- **bool split4Node(TreeNode*)**

  Diese Methode soll in einem Rot-Schwarz-Baum einen Knoten überprüfen, ob dieser ein 4-er Knoten ist. Ein 4-er Knoten im 234-Baum entspricht im Rot-Schwarz-Baum einem schwarzen Knoten, der zwei rote Knoten als Nachfolger hat. Dieser 4-er Knoten soll im Sinne des 2-3-4-Baumes in zwei 2-er Knoten umgewandelt werden. Das bedeutet im Rot-Schwarz-Baum, dass die drei beteiligten Knoten nur umgefärbt werden müssen. Der schwarze Knoten wird dann rot und die vorab roten Knoten werden dann schwarz (s. Abb. 3).

  Der Eingabeparameter dieser Methode sollte der beteiligte schwarze Knoten sein. Die Methode soll generell erst überprüfen, ob es sich um einen 4er Knoten handelt und true zurück geben, wenn dies der Fall ist und die Farben umgefärbt wurden.

- **bool addNode(string, int, double, int)**

  Schreiben Sie eine **iterative Methode**, die einen neuen Knoten **TopDown** in den Rot-Schwarz-Baum einfügt. Die Abbildungen 2 bis 7 zeigen Ihnen die Funktionsweise. Dabei ist folgenderweise vorzugehen:

  1. Traversieren Sie den Baum von der Wurzel bis zum Blatt entsprechend dem einzufügenden Wert. Die Knoten des Suchpfades, die sich im Beispiel beim Einfügen der 26 ergeben, sind in der Abbildung 4 grau markiert. Wird auf dem Suchpfad von der Wurzel bis zum Blatt ein 4-er-Knoten erkannt, wird dieser auf dem Weg nach unten (TopDown) mit der Methode split4Node(TreeNode*) umgefärbt.
  2. Erzeugen Sie einen neuen Knoten mit den entsprechenden einzufügenden Daten und fügen den neuen Knoten als roten Knoten als Nachfolger des Blattes mit entsprechender Suchrichtung ein (s. Abb. 4).
  3. Untersuchen Sie nun die Knoten entlang des Suchpfades vom eingefügten Blattknoten bis zur Wurzel (Bottom-Up), ob zwei rote Knoten aufeinanderfolgen. Ist das der Fall, müssen die notwendigen Rotationen (siehe Abschnitt 3.2.3) durchgeführt werden - verwenden Sie hierfür die Methode balance(TreeNode*). Sie müssen feststellen, in welcher Richtung rotiert werden muss. Beachten Sie, dass ggfls. die Knoten nach der Rotation umgefärbt werden müssen. Die Abbildungen 5 und 6 verdeutlichen hier eine Doppelrotation. Das Ergebnis nach dem Einfügen kann auch als gültiger 2-3-4-Baum interpretiert werden (s.Abb. 7).

- **void printLevelOrder()**

  Diese Methode gibt den Rot-Schwarz-Baum als 2-3-4-Baum auf der Konsole aus. Abbildung 7 zeigt links das Ergebnis nach dem Einfügen von Knoten 26 in den Rot-Schwarz-Baum und rechts das zu interpretierende Ergebnis als 2-3-4-Baum. Die entsprechende Ausgabe in Levelorder visualisiert nur die schwarzen Kanten und wäre dann:

  Niv. 0: (20,25,30)
  Niv. 1: (10) (22) (26,28) (40,50)

Hinweis: Nutzen Sie für die Ausgabe in Levelorder des Rot-Schwarz-Baumes als 2-3-4-Baum zwei Queues. In der einen Queue speichern Sie nur die schwarzen Knoten und in der anderen Queue entsprechend zu dem schwarzen Knoten das Niveau des Knoten im 2-3-4-Baum. Wird ein Knoten aus der Queue entnommen, so entnimmt man auch das dazugehörige Niveau des schwarzen Knotens aus der anderen Queue. Hat der schwarze Knoten rote Nachfolgeknoten, so gehören diese zum schwarzen Knoten (beim 3er oder 4er Knoten bzw. schwarzer Knoten mit 1 roten Nachfolger oder schwarzer Knoten mit 2 roten Nachfolgern) und müssen entsprechend mit ausgegeben werden. Die Nachfolger der roten Knoten sollten dann wieder schwarz sein. Diese nachfolgenden schwarzen Knoten (in der Anzahl 0 bis 4) müssen dann wieder in die Queues mit ihrem entsprechenden 2-3-4-Baum-Niveau gespeichert werden.

- **void printLevelOrder(int niveau)** Implementieren Sie eine Methode, die den Rot-Schwarz-Baum traversiert und nur die Knoten zu einem Niveau im Sinne des 2-3-4-Baumes ausgibt. Die Ausgabe hier für Niveau 1 wäre dann:

  Niv. 1: (10) (22) (26,28) (40,50)

- **void printAll()** Diese Methode gibt den Rot-Schwarz-Baum als BST in Levelorder auf der Konsole aus. Verwenden Sie hierfür nur 1 Queue. Das Niveau muss nicht ausgegeben werden. Einen Beispiel-Output aus den ersten Test-Cases finden Sie im Auszug aus den Testläufen (Zeilen 23-32).

- **int proofRBCriterion(Treenode*)** Implementieren Sie eine rekursive Methode, die für den gegebenen Rot-Schwarz-Baum das Kriterium der „Schwarzausgeglichenheit“ überprüft und die Anzahl der schwarzen Kanten zurück gibt. Ein Rot-Schwarz-Baum heißt „schwarzausgeglichen“, wenn die Anzahl der schwarzen Kanten vom Blattknoten bis zur Wurzel für alle dieser Pfade gleich ist.

  Für den gegebenen Rot-Schwarz-Baum wäre dies gültig und hätte die Höhe eins.

  Hinweis: Schreiben Sie eine **rekursive Bottom-Up-Methode** (im Sinne der Postorder), die die Höhe (=Anzahl der schwarzen Kanten) der Knoten im Sinne des 2-3-4-Baumes bestimmt. Pro Knoten sind dann alle Fälle zu überprüfen:

  1. beide Nachfolger existieren nicht: dann ist ein Blattknoten gefunden, die Höhe 0 wird zurück gegeben.
  2. beide Nachfolger rot: Die Höhen der roten Knoten müssen übereinstimmen mit der Höhe des schwarzen Vorgängers.
  3. einer der Nachfolger ist rot, der andere schwarz: die Höhe des roten Knotens muss um eins höher sein als die Höhe des schwarzen Nachfolgeknotens und ist identisch zur Höhe des schwarzen Vorgängerknotens.
  4. beide Nachfolger sind schwarz: die Höhen der schwarzen Knoten müssen gleich sein und es wird die Höhe der schwarzen Nachfolgerknoten + 1 zurück gegeben.

5. Falls einer der Nachfolger nicht existiert, wird die Höhe des anderen Knotens übernommen: beim schwarzen Nachfolgeknoten die Höhe + 1, beim roten Nachfolgeknoten wird die gleiche Höhe zurück gegegben.
6. Wird ein Fehler erkannt, soll -1 zurück gegeben werden und die rekursive Methode abgebrochen werden.

Implementieren Sie zu der rekursiven Methode auch eine öffentliche (public) Starterfunktion, die die rekursive Methode aufruft und die Höhe der Wurzel ausgibt bzw. eine Fehlermeldung, wenn der Baum nicht schwarzausgeglichen ist.

Hinweis: Sie können während Ihrer Tests manuell Knoten umfärben, indem sie per friend-Methode TreeNode get_anker(Tree&) direkten Zugriff auf die Knoten eines Baumes bekommen und das red Attribut per Setter verändern.

- **rotateTreeRight()/rotateTreeLeft()**

  Es wird der Methode jeweils der untere rote Knoten übergeben. Den Vorgängerknoten, der entweder rot oder schwarz sein kann, erreicht man über die getParent()-Methode. Führen Sie die jeweilige Rotation durch und beachten Sie die richtige Einfärbung der rotierten Knoten. Beachten Sie außerdem, dass der Vorgängerknoten vom oberen Knoten auf den neuen rotierten Knoten verweist. Führt man diese Methoden hintereinander aus, erhält man eine Doppelrotation. Ggfls. können Sie auch dafür eigene Methoden zur Durchführung der Links-Rechts- sowie der Rechts-Links-Rotation implementieren. Geben Sie jeweils auf der Konsole aus, welche Rotation mit welchen Knoten durchgeführt wurde. Beispiel: LR(20,25) (Linksrotation zwischen Knoten 20 und 25) oder RR(30,25) (Rechtsrotation zwischen Knoten 30 und 25) oder DR(28,10,15) (Doppelrotation zwischen Knoten 28,10 und 15).

  Hinweis: Visualisieren Sie die jeweiligen Rotationen mit ihren Vorgängerknoten mit dem Zustand vor und nach ihren notwendigen Rotationen.

Testläufe:

```
============================   // Beispiel: Menü der Anwendung
ADS-Rot-Schwarz-Baum Praktikum
============================
1) Datensatz einfuegen, manuell
2) Datensatz einfuegen, CSV Datei
3) Suchen
4) Ausgabe in Levelorder
5) Ausgabe in Levelorder (mit Niveauauswahl)
?> 1                // Beispiel: manuelles Hinzufügen eines Datensatzes

+ Bitte geben Sie die den Datensatz ein
Name ?> Mustermann
Alter ?> 1
```

```
Einkommen ?> 1000.00
PLZ ?> 1
+ Ihr Datensatz wurde eingefügt

[...]               // Hier Eingabe weiterer Datensätze

?> 5                // Beispiel: Anzeigen eines Trees in Levelorder Einträgen

Ausgabe in Levelorder als binärer Suchbaum:

ID | Name       | Alter | Einkommen |  PLZ  | Pos   | Red
---+------------+-------+-----------+-------+-------+-------
1  |      Ritter|      1|       2000|      1|   2002| 0
4  |     Schmitt|      1|        500|      2|    503| 1
2  |      Kaiser|      1|       3000|      1|   3002| 0
3  |        Hans|      1|        500|      1|    502| 0
0  |  Mustermann|      1|       1000|      1|   1002| 0
5  |     Schmitz|      1|        400|      2|    403| 1

Ausgabe in Levelorder als 2-3-4-Baum:
Niv. 0: (503, 2002)
Niv. 1: (403, 502) (1002) (3002)
```

3.2 Lösungshinweise

Beachten Sie ALLE Lösungshinweise und verstehen Sie zunächst wie ein Rot-Schwarz-Baum funktioniert.

3.2.1 Rot-Schwarz-Baum Kriterien

Die Kriterien eines Rot-Schwarz-Baumes müssen zu jeder Zeit alle erfüllt sein!

1. Jeder Knoten ist entweder rot oder schwarz.
2. Jeder neu einzufügende Blattknoten ist rot.
3. Die Kinder von einem roten Knoten sind schwarz.
4. Es gibt keine zwei aufeinanderfolgende rote Knoten.
5. Kriterium für Schwarz-Ausgeglichenheit:

   Für jeden Knoten k gilt: Jeder Pfad von k zu einem Blatt enthält die gleiche Anzahl schwarzer Knoten.

6. Die Wurzel ist immer schwarz.

3.2.2 Top-Down Einfügen

Beim Top-Down Einfügen werden entlang des Suchpfades bis zur Einfügeposition alle 4er-Knoten, das sind die schwarzen Knoten mit 2 roten Nachfolgern, umgefärbt. In Abbildung 8 wird dies für den Knoten n mit seinen Kindern k und p durchgeführt.

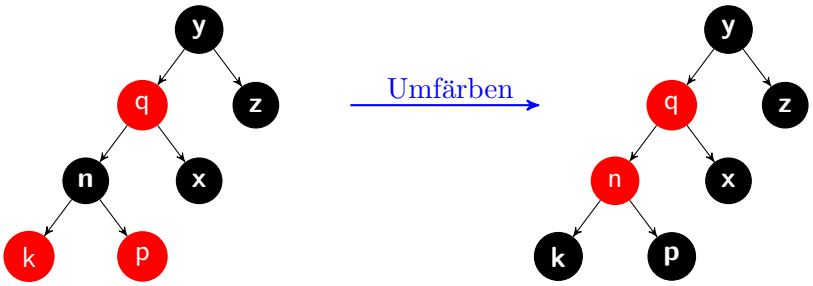

Abbildung 8: Umfärben von Knoten (k, n, p)

Dabei kann es zu einer Verletzung des 4. Kriterums von Rot-Schwarz-Bäumen kommen. Dies muss nach dem Umfärben geprüft und ggf. durch Rotationen behoben werden.

3.2.3 Rotationen

Durch Einfügen neuer Knoten oder Umfärben können zwei aufeinander folgende Knoten rot eingefärbt werden. Dies verletzt das 4. Kriterium für einen Rot-Schwarz-Baum und der Baum muss durch Rotationen ausgeglichen werden. Möglich sind Rechts-, Links- oder Doppelrotationen (rechts-links oder links-rechts).

Abbildung 9 zeigt die vier möglichen Rotationen und das jeweilige Ergebnis im Überblick.

Grundsätzlich kann man sich merken:

- Die Richtungen entlang des Suchpfades bis zur Einfügeposition, in der die Knoten verbunden sind, geben die Rotationen vor. Damit ist die Verbindung vom Parent zum rechten oder linken Nachfolger gemeint.
- Zeigen beide aufeinander folgende rote Knoten in die gleiche Richtung, so reicht eine einfache Rotation.
- Zeigen die aufeinander folgenden roten Knoten in verschiedene Richtungen, so muss mit einer ersten Rotation der untere Knoten an den oberen angeglichen und danach mit einer zweiten Rotation der schwarze Knoten mit dem roten Knoten rotiert werden. Dies bewirkt einen ausgeglichenen Baum.

Abbildung 10 zeigt die Knoten 2 und 4 in rot hintereinander. Da diese nicht aus der gleichen Richtung kommen, muss zunächst eine Linksrotation der Knoten 2 und 4 durchgeführt werden.

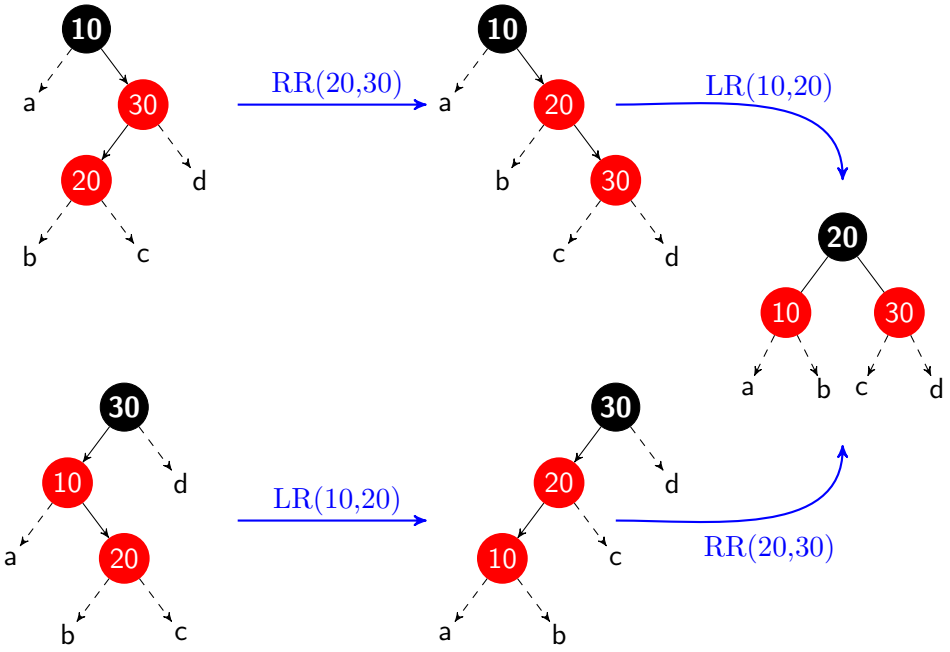

Abbildung 9: Möglichen Rotationen im Rot-Schwarz-Baum.

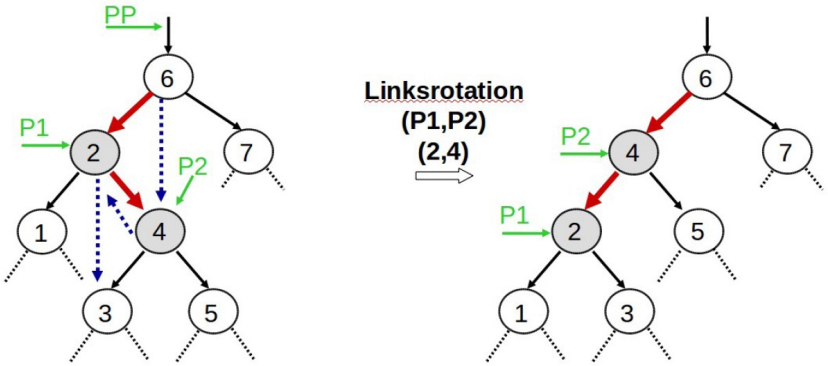

Abbildung 10: Linksrotation der Knoten 2 und 4

Im Anschluss daran wird die zweite Rotation ausgeführt, wie in Abbildung 11 zu sehen. Sie zeigt eine Rechtsrotation der Knoten 6 und 4. Nach dieser Rotation sind die Knoten 2 und 6 rot und die Wurzel 4 ist schwarz. Beachten Sie daher, dass ggfs. die Farben der an der Rotation beteiligten Knoten geändert werden müsse..

Eine Doppelrotation setzt sich aus diesen beiden Rotationen zusammen. Die Abbildungen 10 und 11 ergeben zusammen eine Doppelrotation in Links-Rechts Reihenfolge. Daher können Sie in Ihrem Programm die beiden Rotationen hintereinander ausführen um eine Doppelrotation zu erzeugen.

Achtung: Kontrollieren Sie das richtige Einfärben der Knoten nach Einzel- und Doppelrotationen!

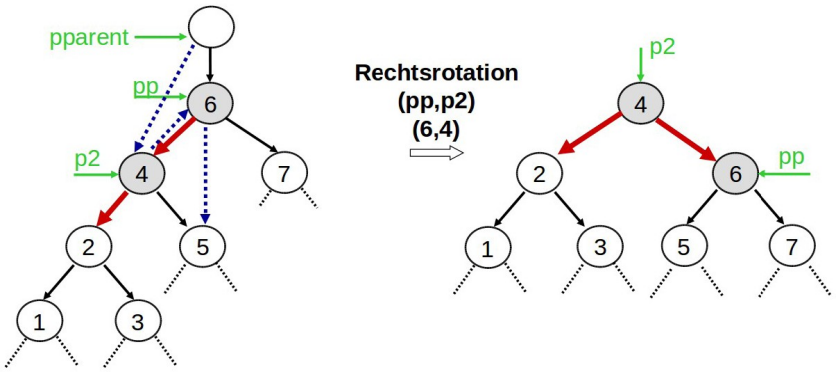

Abbildung 11: Rechtsrotation der Knoten 6 und 4

3.2.4 Alte Methoden

Neben den neu zu implementierenden Methoden müssen Sie auch in einigen alten Methoden Anpassungen vornehmen. Folgendes **muss** beachtet werden:

- Ihre **Einfügemethode** muss **iterativ** programmiert werden. Haben Sie eine rekursive Einfügemethode verwendet, passen Sie bitte ihre Implementierung an!
- Beachten Sie, dass die Klasse Treenode das Attribut Treenode* parent zusätzlich erhalten hat. Passen Sie ggfls. ihre Implementierung an.
- Beim Einfügen wird das Red-Flag des neuen Knotens grundsätzlich auf *true* gesetzt und der Baum muss auf die Rot-Schwarz-Kriterien geprüft werden.
- Das Löschen von Knoten wird in diesem Praktikum nicht implementiert. Da es in einem Rot-Schwarz-Baum nicht trivial ist, wird dies hier nicht weiter verwendet oder angepasst.
- In der Suche brauchen Sie im Vergleich zum BST keine Anpassungen vornehmen.