Bachelorarbeit

Andreas Windorfer

10. Mai 2020

Zusammenfassung

Inhaltsverzeichnis

1 Fazit **4**

2 Rot-Schwarz-Baum **4**

2.1 Grundoperationen 5

2.2 Tango-Baum konformes vereinigen 15

2.3 Tango-Baum konformes aufteilen 19

3 dd **20**

3.1 Einführung der amortisierten Laufzeitanalyse 20

1 Fazit

2 Rot-Schwarz-Baum

Der Rot-Schwarz-Baum gehört zur Gruppe der **balancierten BST** und erfüllt alle Eigenschaften um ihn als Hilfsstruktur im Tango-Baum zu verwenden. Genau das ist auch die Rolle des Rot-Schwarz-Baumes in dieser Ausarbeitung. Bei balancierten BST gilt für die Höhe $h = O(n)$, mit $n =$ Anzahl der Knoten. Jeder Knoten benötigt ein zusätzliches Attribut, um eine Farbinformation zu speichern. Der Name der Datenstruktur kommt daher, dass die beiden durch das zusätzliche Attribut unterschiedenen Zustände als *Rot* und *Schwarz* bezeichnet werden. Die Farbe ist also eine Eigenschaft der Knoten und im folgenden wird einfach von roten bzw. schwarzen Knoten gesprochen. Innerhalb mancher Operationen wird von einem Knoten aus direkt auf dessen Vater zugegriffen, so dass man sich im Baum auch nach oben hin bewegen kann. In Implementierungen wird das so umgesetzt, dass es zusätzlich zu den beiden Zeigern auf die Kinder noch einen zum Vater gibt. Als Blätter werden schwarze Sonderknoten verwendet, deren Schlüssel auf einen Wert außerhalb des Universums, hier *null*, gesetzt wird, um sie eindeutig erkennen zu können. gehört per Definition nicht zur Schlüsselmenge des RBT. Fehlende Kinder von Knoten mit gewöhnlichem Schlüssel werden durch solche Blätter ersetzt. Folgende zusätzliche Eigenschaften müssen bei einem Rot-Schwarz-Baum erfüllt sein.

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes Blatt (Sonderknoten) ist schwarz.
4. Der Vater eines roten Knotens ist schwarz.
5. Für jeden Knoten gilt, dass alle Pfade, die an ihm starten und an einem Blatt (Sonderknoten) enden, die gleiche Anzahl an schwarzen Knoten enthalten.

Sei $(v_0, v_1, ..., v_n)$ ein Pfad von einem Knoten v_0 zu einem Blatt v_n. Die Anzahl der schwarzen Knoten innerhalb $(v_1, ..., v_n)$ wird als **Schwarz-Höhe** $bh(v_0)$ von Knoten v_0 bezeichnet. Die eigene Farbe des betrachteten Knotens bleibt dabei also außen vor. Dadurch hat ein Knoten die gleiche Schwarz-Höhe wie ein rotes Kind und eine um eins erhöhte Schwarz-Höhe gegenüber einem schwarzen Kind. Die Schwarz-Höhe der Wurzel entspricht der

Schwarz-Höhe des Baumes $bh(T)$, wobei ein leerer Baum Schwarz-Höhe 0 hat. Die Schwarz-Höhe eines Knoten x ist genau dann eindeutig wenn er Eigenschaft 5 nicht verletzt. Hält x Eigenschaft 5 ein und sei i die Anzahl schwarzer Knoten in den entsprechenden Pfaden, so gilt $bh(x) = i$ wenn x rot ist und $bh(x) = i - 1$ wenn x schwarz ist. Ist $bh(x)$ eindeutig, so enthält jeder Pfad der mit x startet und an einem Blatt endet $bh(x) + 1$ schwarze Knoten, wenn x schwarz ist und $bh(x)$ schwarze Knoten wenn x rot ist. Jeder Knoten speichert seine Schwarz-Höhe als weiteres Attribut, da wir dieses in Abschnitt 2.2 benötigen. Natürlich muss das Attribut, dann auch gesetzt und gepflegt werden, wobei es bei Sonderknoten fest mit 0 belegt ist. Die Im folgenden wird **RBT** (Red-Black-Tree) als Abkürzung für Rot-Schwarz-Baum verwendet. Aufgrund der Sonderknoten gibt es eine etwas spezielle Situation, bei einem RBT mit Höhe 1. Diese Konstellation ist nur mit einem einzelnen Sonderknoten erreichbar, so dass man statt dessen auch einfach den leeren Baum verwenden könnte. Auch diese Konstellation erfüllt aber die Eigenschaften, so dass sie kein Problem darstellt.

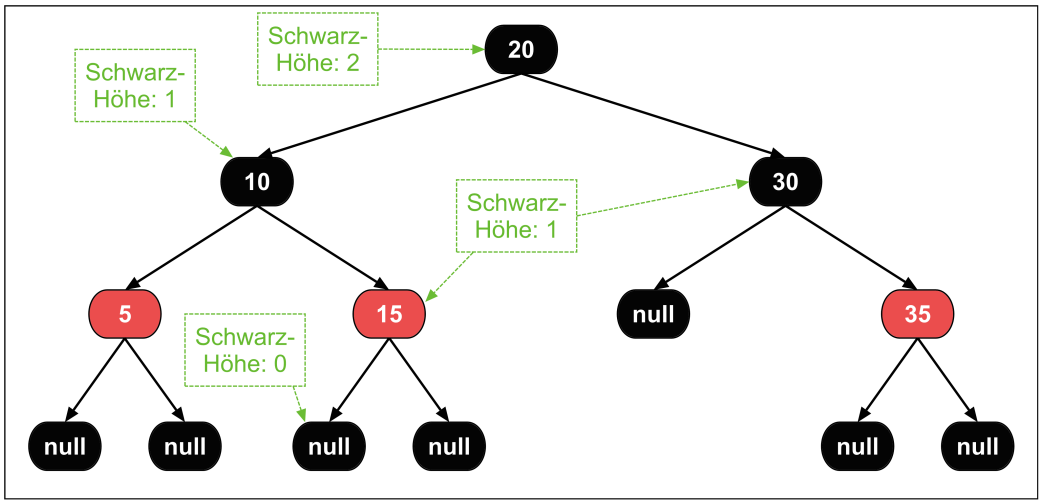

Abbildung 1: Rot-Schwarz-Baum ohne Verletzung von Eigenschaften.

2.1 Grundoperationen

Suchen im Rot-Schwarz-Baum Die Suche unterscheidet sich nur in einem Punkt von der in **??** vorgestellten. Wird nach einem Schlüssel gesucht, der im RBT nicht vorhanden ist, so wird einer der Sonderknoten erreicht. In diesem Fall wird die Suche abgebrochen und eine leere Referenz zurückzugeben. Die Operation verändert den RBT nicht.

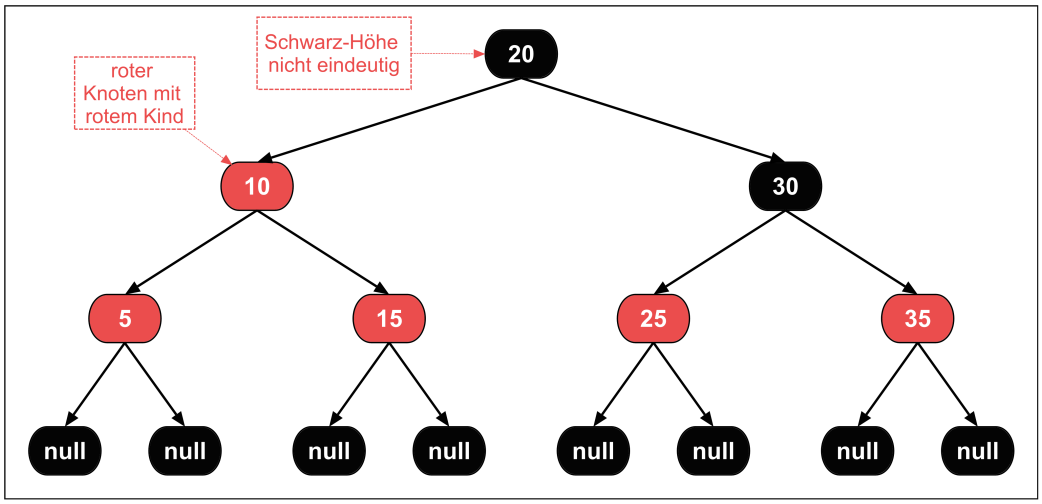

Abbildung 2: Rot-Schwarz-Baum bei dem Eigenschaft vier und fünf verletzt sind.

Einfügen in den Rot-Schwarz-Baum Sei k der einzufügende Schlüssel. Zunächst wird wie beim Suchen vorgegangen. Wird k gefunden wird der RBT nicht verändert. Ansonsten wird ein Sonderknoten b erreicht. Ein neu erzeugter roter Knoten v_k mit Schlüssel k und Schwarz-Höhe 1 nimmt den Platz von b ein. k werden Sonderknoten als linkes und rechtes Kind angefügt. k ist nun im Baum enthalten, es muss jedoch auf mögliche Verletzungen der fünf Eigenschaften geachtet werden. Welche können betroffen sein ?

1. Es ist immer noch jeder Knoten entweder rot oder schwarz.
2. Wurde in den leeren Baum eingefügt, so ist der neu eingefügte rote Knoten die Wurzel, was eine Verletzung darstellt. Waren bereits Knoten im Baum vorhanden blieb die Wurzel unverändert.
3. Aufgrund der Sonderknoten sind die Blätter immer noch schwarz.
4. Der Baum wird nur direkt an der Einfügestelle verändert. Der neue Knoten hat schwarze Kindknoten, er könnte jedoch einen roten Vater haben, so dass diese Eigenschaft verletzt wäre.
5. Die Schwarz-Höhe von v_k ist korrekt gesetzt. Die Schwarz-Höhe keines anderen Knotens hat sich verändert, denn den Platz eines schwarzen Knoten mit Schwarz-Höhe 0 nimmt nun ein roter Knoten mit Schwarz-Höhe 1 ein. Eigenschaft fünf bleibt also erhalten.

Es können also die Eigenschaften zwei und vier betroffen sein. Jedoch nur eine von ihnen, denn Eigenschaft zwei wird genau dann verletzt wenn der

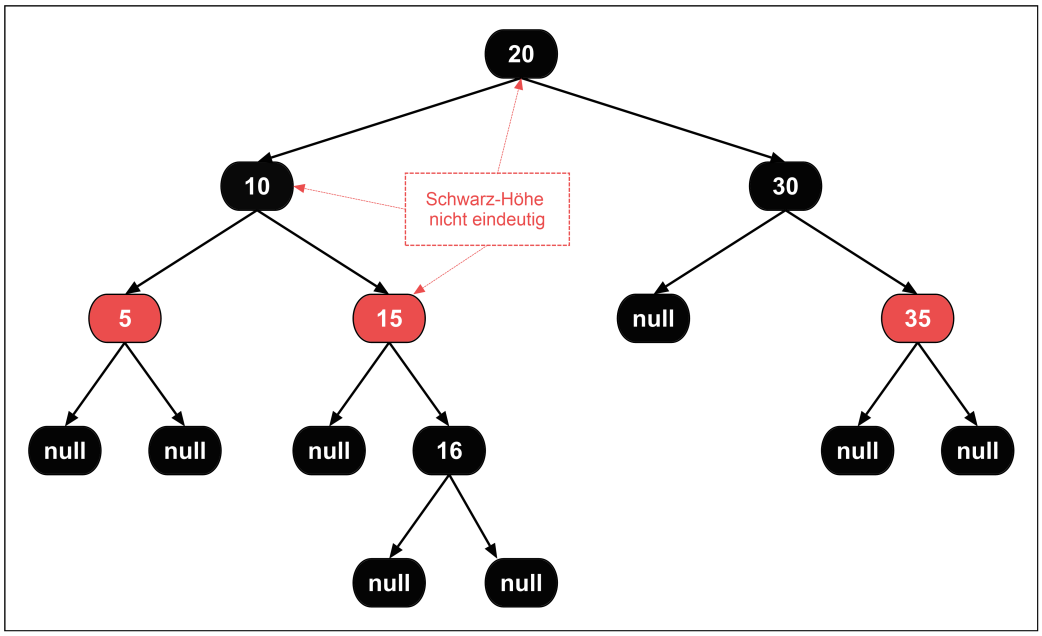

Abbildung 3: Rot-Schwarz-Baum bei dem Eigenschaft fünf verletzt ist.

neue Knoten die Wurzel des Baumes ist, dann kann er aber keinen roten Vater haben.
Zur Korrektur wird zum Ende von Einfügen eine zusätzliche Operation, **einfügen-fixup(Knoten** v_{in}**)** aufgerufen. Diese Operation arbeitet sich von v_i startend, solange in einer Schleife nach oben im BST durch, bis alle Eigenschaften wieder erfüllt sind. Die Schleifenbedingung ist, dass eine Verletzung vorliegt. Dazu muss geprüft werden ob der betrachtete Knoten x die rote Wurzel des Gesamtbaumes ist, oder ob er und sein Vater beide rot sind. Vor dm ersten Durchlauf wird $x = v_{in}$ gesetzt. Innerhalb der Schleife werden sechs Fälle unterschieden. Im folgenden wird auf vier Fälle detailliert eingegangen. Die restlichen zwei verhalten sich symmetrisch zu einem solchen. Jeder der Fälle verantwortet, dass zum Start der nächsten Iteration wieder nur maximal eine der beiden Eigenschaften zwei oder vier verletzt sein kann und Eigenschaft vier höchstens an einem Knoten verletzt ist. Die Fallauswertung geschieht in aufsteigender Reihenfolge. Deshalb kann man innerhalb einer Fallbehandlung verwenden, dass die vorherigen Fallbedingungen nicht erfüllt sind. Eigenschaft eins bleibt in der Beschreibung außen vor, da es während der gesamten Ausführungszeit der Operation nur Knoten gibt, die entweder rot oder schwarz sind.

Fall 1: x **ist die rote Wurzel des RBT:** Dieser Fall wird behandelt in

dem man die Wurzel schwarz färbt. Man muss noch zeigen, dass es durch das Umfärben zu keiner anderen Verletzung gekommen ist.

Betrachtung der Eigenschaften:

1. -
2. Die Wurzel wurde schwarz gefärbt.
3. Die Blätter (Sonderknoten) sind unverändert.
4. Es wurden weder rote Knoten hinzugefügt, noch wurde die Kantenmenge verändert.
5. Das Umfärben der Wurzel kann hierauf keinen Einfluss haben, da sie in der Berechnung der Schwarz-Höhe jedes Knotens außen vor ist.

Es wird also keine Eigenschaft mehr verletzt und die Schleife wird keine weitere Iteration durchführen.
Die Fälle 2 - 6 behandeln nun die Situation, dass sowohl x als auch dessen Vater y rote Knoten sind. Da Eigenschaft fünf nach jeder Iteration erfüllt ist muss y einen Bruder haben. Denn da zu Beginn einer Iteration nur eine Eigenschaft verletzt sein kann, kann der rote y nicht die Wurzel sein, also muss auch y einen Vorgänger z haben. Da z kein Blatt(Sonderknoten) ist, müssen beide Kinder vorhanden sein. Außerdem muss z schwarz sein, ansonsten wäre Eigenschaft vier an zwei Knoten verletzt.

Fall 2: y hat einen roten Bruder: Diesen Fall veranschaulicht Abbildung 5. Da y rot ist, muss z schwarz sein, ansonsten wäre Eigenschaft vier mehrfach verletzt gewesen. Nun wird z rot gefärbt und beide Kinder von z, also y und dessen Bruder, schwarz. Die Schwarz-Höhe von z wird um eins erhöht. Somit ist der Vater von x nun schwarz und die Verletzung der Eigenschaft vier wurde an dieser Stelle behoben. Wie sieht es aber mit den Verletzungen insgesamt aus ?

Betrachtung der Eigenschaften:

1. -
2. Wenn z die Wurzel des Baumes ist, wurde sie rot gefärbt und eine Verletzung liegt vor.
3. Der rot umgefärbte Knoten z' hat zwei Kinder, somit wurde kein Blatt rot gefärbt.

4. Wenn der rot gefärbte Knoten z' nicht die Wurzel ist, könnte er einen roten Vater haben und Eigenschaft vier ist weiterhin Verletzt. Das Problem liegt nun aber zwei Baumebenen höher.

5. Die Schwarz-Höhen der Vorfahren von z' bleiben unverändert, da jeder Pfad von ihnen zu einem Blatt auch entweder y' oder dessen Bruder enthält. z' Schwarz-Höhe steigt um eins gegenüber z, bleibt aber eindeutig. An keinem anderen Knoten ändert sich die Schwarz-Höhe.

Es kann also wieder nur entweder Eigenschaft zwei oder vier verletzt sein. Wenn das Problem noch nicht an der Wurzel ist, liegt es zumindest zwei Ebenen näher daran. x wird auf z' gesetzt.

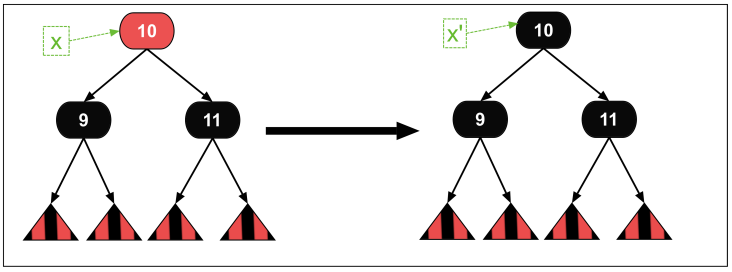

Abbildung 4: einfügen-fixup. Dargestellt ist Fall 1

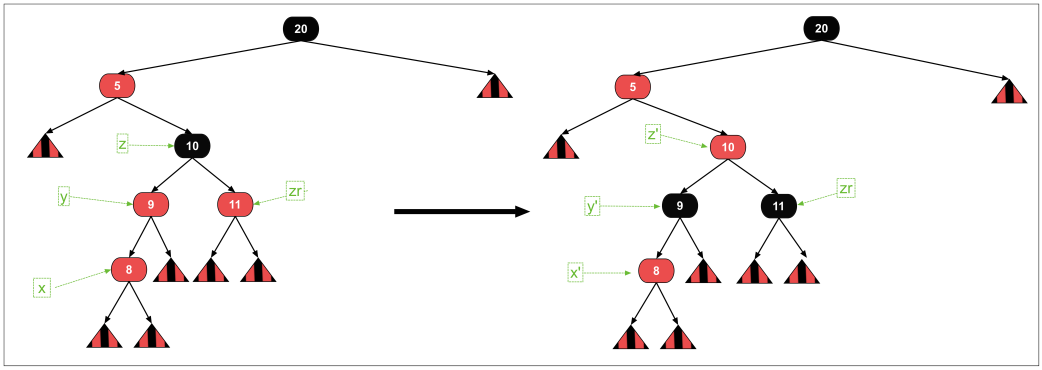

Abbildung 5: einfügen-fixup. Dargestellt ist Fall 2

Fall 3: x ist ein linkes Kind. y ist ein linkes Kind:
Abbildung 6 zeigt eine entsprechende Situation. Es wird eine Rechtsrotation auf y ausgeführt. Anschließend wird z rot gefärbt und y schwarz.

Betrachtung der Eigenschaften:
Dazu werden vier weitere Variablen auf Knoten verwendet. Es zeigt auf xl das

linke Kind von x, xr entsprechend das rechte Kind. yr und zr bezeichnen die rechten Kinder von y bzw. z. Nachfolgend wird verwendet, dass die Teilbäume mit den Wurzeln xl, xr, yr und zr durch die Ausführung unverändert bleiben.

1. -

2. Wenn z zu Beginn nicht die Wurzel des Gesamtbaumes war, bleibt diese unverändert. Ansonsten wurde durch die Rotation y' zur neuen Wurzel und y' wurde schwarz gefärbt.

3. Alle vier Plätze in der zweiten Ebene unter z' werden von den unveränderten Teilbäumen mit den Wurzeln xl', xr', yr' oder zr' besetzt. An den Blättern verändert sich also durch die Ausführung nichts.

4. Knoten x' ist linkes Kind des schwarzen y'. x' Teilbäume blieben unverändert. Der linke Teilbaum von y' enthält somit keine aufeinanderfolgenden roten Knoten. Das rechte Kind von y' ist der rote Knoten z'. Rechts an z' hängt nun ein unveränderter Teilbaum, dessen Wurzel zuvor Bruder von y war. Dieser ist nach Fallunterscheidung ein schwarzer Knoten. Links hängt ebenfalls ein unveränderter Teilbaum, dessen Wurzel zuvor rechter Nachfolger von y war. Der rechte Nachfolger von y muss schwarz sein, ansonsten wäre Eigenschaft vier an zwei Knoten verletzt gewesen. Im Teilbaum mit Wurzel y gibt es also keine aufeinanderfolgenden roten Knoten. Da y' schwarz gefärbt wurde, kann auch außerhalb des Teilbaumes mit y' keine neue Verletzung entstanden sein.

5. Es gilt $bh(xl) = bh(xr) = bh(yr) = bh(zr) = bh(z) - 1$. Wie oben bereits erwähnt wird die zweite Ebene unter der Wurzel y' von den unveränderten Teilbäumen xl', xr', yr' und zr' gebildet. Es müssen also lediglich die Knoten x', y' und z' betrachtet werden. An x' und an z' folgen schwarze Knoten mit der Schwarz-Höhe $bh(z) - 1$. Die Schwarz-Höhen von x' und z' sind also eindeutig und es gilt $bh(x') = bh(z') = bh(z)$. y' Kinder sind die roten Knoten x' und z'. Da beide Kinder rot sind gilt $bh(y') = mathitbh(x') = bh(z)$. Somit sind alle Schwarz-Höhen im betrachteten Teilbaum eindeutig. Die neue Wurzel der Teilbaumes y' hat die gleiche Schwarz-Höhe und die gleiche Farbe wie die vorherige Wurzel z. Damit kann es auch im Gesamtbaum zu keiner Verletzung der Eigenschaft gekommen sein.

Es ist keine der Eigenschaften verletzt, daher wird es zu keiner Iteration mehr kommen.
Fall 4: x ist ein rechtes Kind. y ist ein linkes Kind.:
Dieser in Abbildung 7 gezeigte Fall wird so umgeformt, dass eine Situation

entsteht bei der Fall drei angewendet werden kann. Dazu wird eine Linksrotation an Knoten x durchgeführt.

Betrachtung der Eigenschaften:
Zu Veränderungen kommt es durch die Rotation lediglich im linken Teilbaum von z. Es sei xl das linke Kind von x, und xr das rechte Kind von x. yl ist das linke Kind von y. xl, xr und yl müssen schwarz sein, ansonsten wäre Eigenschaft vier mehrfach verletzt gewesen.

1. -

2. Die Wurzel bleibt unverändert.

3. Die Teilbäume mit den Wurzeln xl, xr und yl enthalten alle Blätter innerhalb des linken Teilbaumes von z. Die Teilbäume xl, xr und yl bleiben durch die Rotation unverändert und xl', xr' und yl' enthalten auch alle Blätter des linken Teilbaumes von z'.

4. Da x und y rot sind müssen z, xl und xr schwarz sein. Nach der Rotation ist y' linkes Kind von x'. x' ist Kind vom schwarzen z'. Alle verbleibenden Kinder von x' und y' werden durch die unveränderten Teilbäume xl', xr' und yl' gebildet. Deren Wurzeln müssen schwarz sein, ansonsten hätte es in ursprünglichen Baum an mehr als einem Knoten eine Verletzung von Eigenschaft vier gegeben. Durch die Rotation verbleibt es also bei einer Verletzung der Eigenschaft vier in der gleiche Baumebene. Die beiden beteiligten roten Knoten sind nun aber jeweils linke Kinder.

5. $bh(yl) = bh(xl) = bh(xr) = bh(yl') = bh(xl') = bh(xr')$. Die Schwarz-Höhen von x und y bleiben unverändert. Damit kommt es auch bei z zu keiner Veränderung bei der Schwarz-Höhe.

Es sind also weiterhin zwei rote aufeinanderfolgende rote Knoten in den gleichen Baumebenen vorhanden. Diese sind nun aber beides linke Kinder. Der Bruder des oberen roten Knotens ist der selbe schwarze Knoten wie vor der Ausführung von Fall 4. Damit kann direkt mit dem bearbeiten von Fall 3 begonnen werden.

Fall 5: x ist ein rechtes Kind. y ist ein rechtes Kind:
Abbildung 8 zeigt den zu Fall 3 Links/Rechts-Symmetrischen Fall 5.
Fall 6: x ist ein linkes Kind.y ist ein rechtes Kind:
Abbildung 9 zeigt den zu Fall 4 Links/Rechts-Symmetrischen Fall 6.

Sei h die Höhe des Gesamtbaumes vor Aufruf von einfügen-Fixup. Fall 2 kann maximal $\frac{h}{2}$ mal ausgewählt werden, bevor x oder y an der Wurzel liegt. Nach einer Iteration bei der nicht Fall 2 ausgewählt wird, terminiert einfügen-fixup. Der Aufwand in jedem Fall ist unabhängig von n. Für die Laufzeit gilt deshalb $O(h)$.

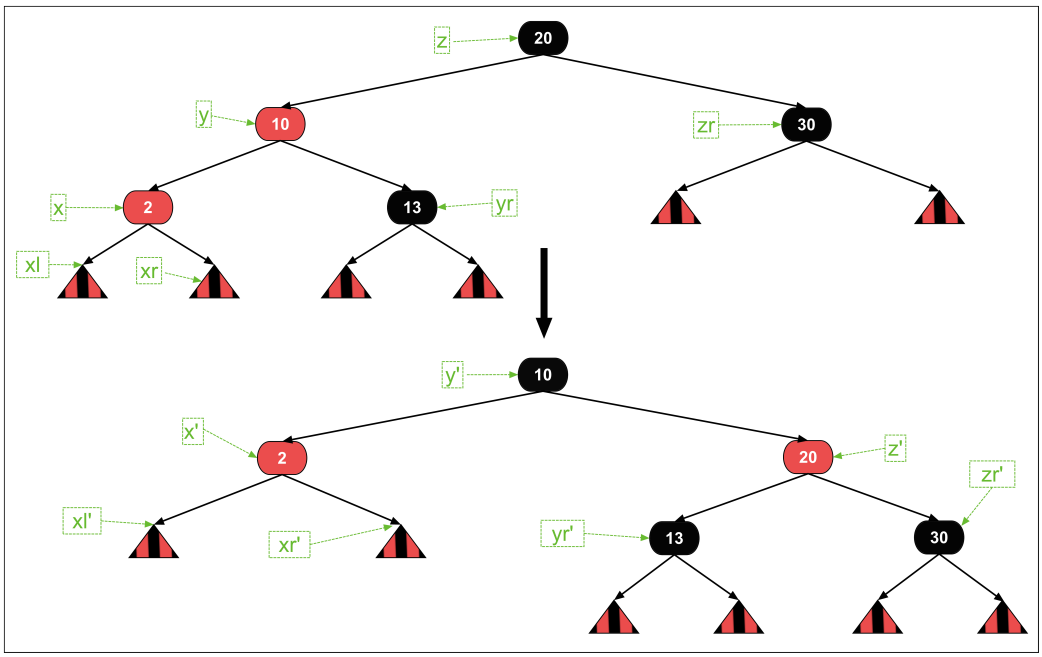

Abbildung 6: einfügen-fixup. Dargestellt ist Fall 3

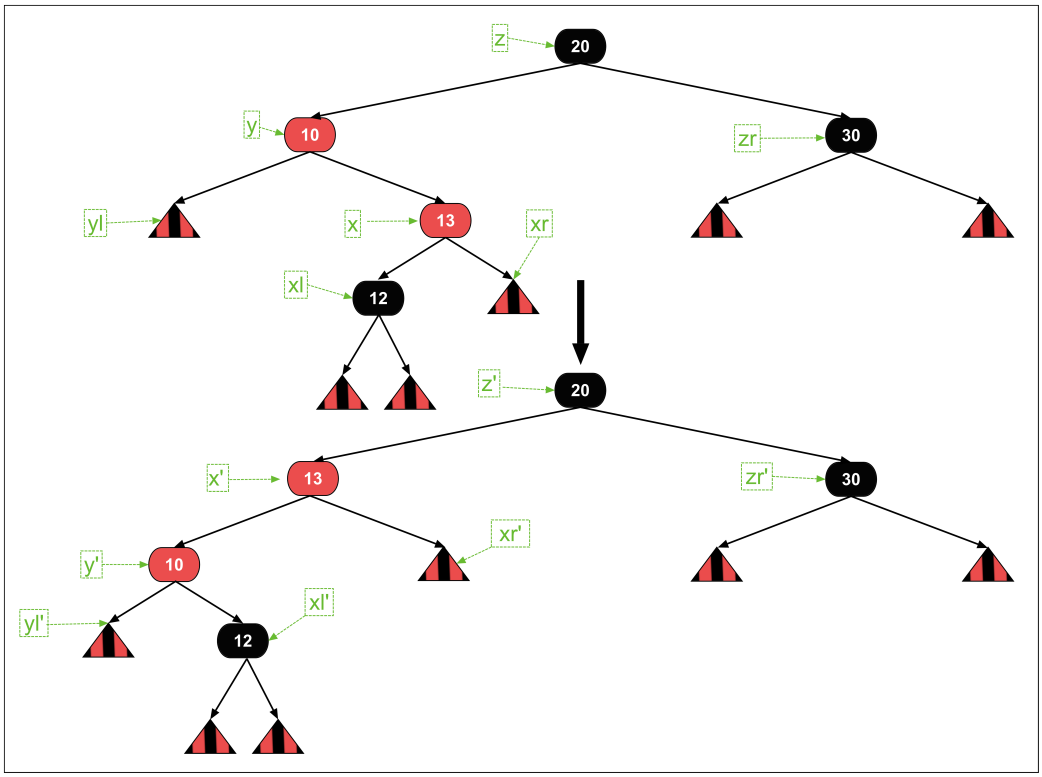

Abbildung 7: einfügen-fixup. Dargestellt ist Fall 4

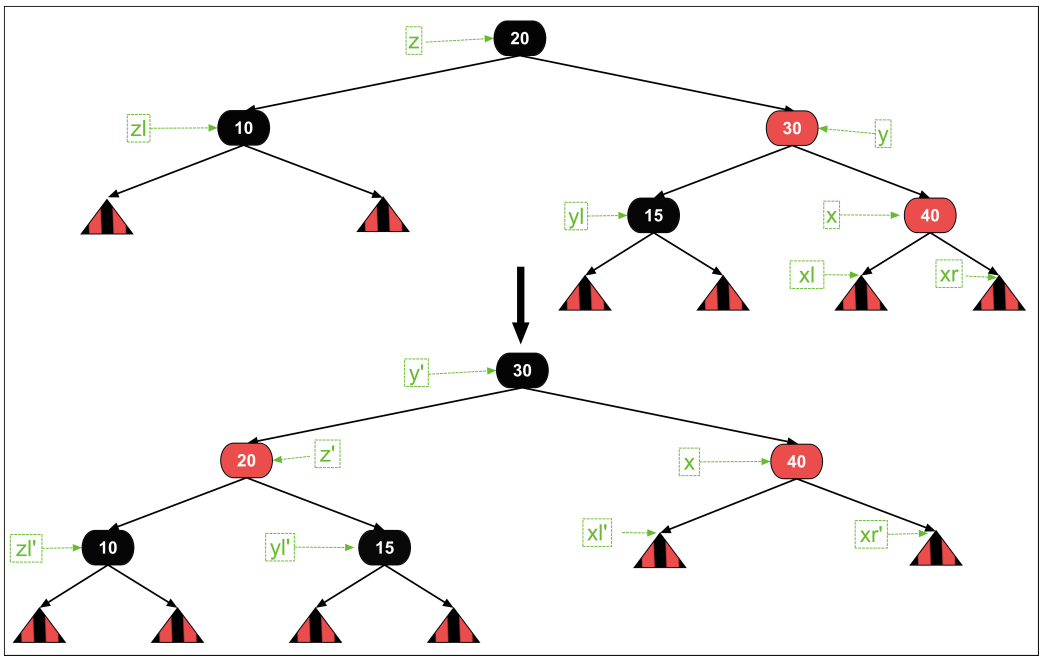

Abbildung 8: einfügen-fixup. Dargestellt ist Fall 5

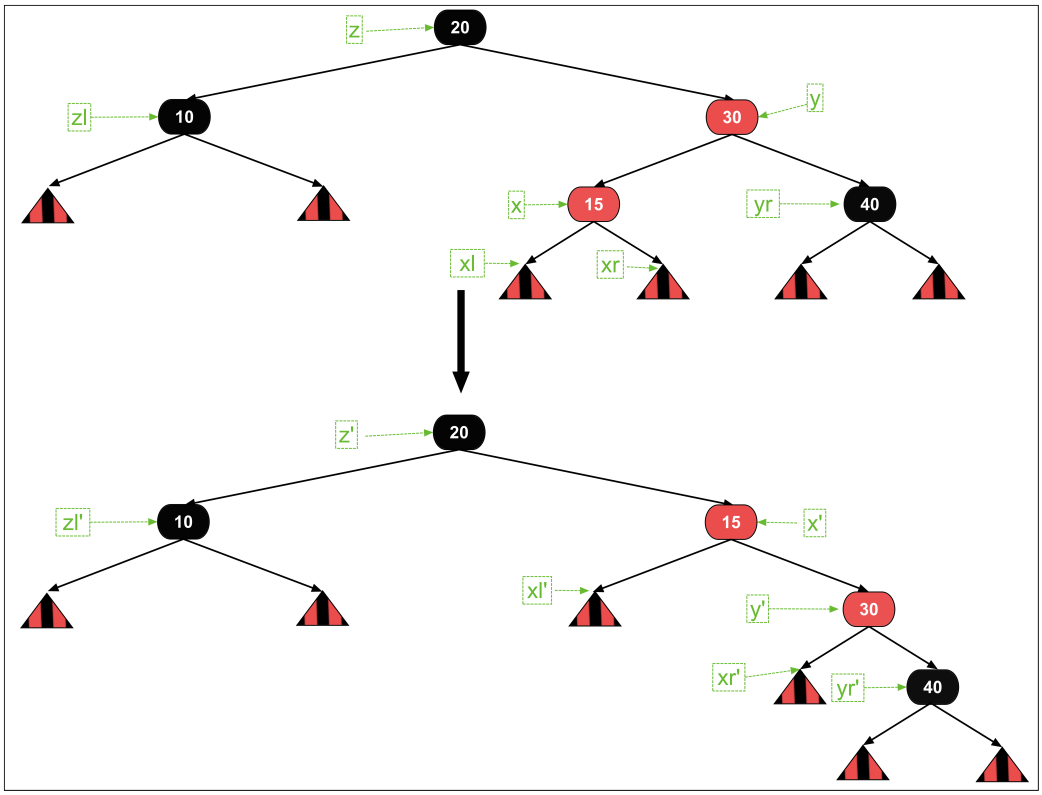

Abbildung 9: einfügen-fixup. Dargestellt ist Fall 6

Löschen aus dem Rot-Schwarz-Baum Die Reparatur des RBT nach dem entfernen eines Knotens ist aufwendiger als, die nach dem einfügen. Da der RBT in der Rolle als Hilfsstruktur für den Tango-Baum keine solche Operation benötigt entfällt, die Beschreibung. In [1] ist eine detaillierte Beschreibung enthalten.

Laufzeit der Grundoperationen Zu Beginn des Kapitels wurde erwähnt, dass für die Höhe h eines RBT mit n Knoten $h = O(\log n)$ gilt. Das wird nun gezeigt.

Lemma 2.1. *Für die Höhe h eines RBT T mit n Knoten gilt $h = O(\log n)$*
.

Beweis. Sei w die Wurzel von T und m die Anzahl der inneren Knoten von T. Zunächst wird gezeigt, dass T mindestens $2^{bh(w)} - 1$ innere Knoten enthält. Dies geschieht mit Induktion über h. Für $h = 0$ bzw. $h = 1$ mit $2^0 - 1 = 0$ stimmt die Behauptung, denn der Baum ist leer oder ein einzelner Sonderknoten. Induktionsschritt mit Höhe $h + 1$:

Sei *tl* der linke Teilbaum von w und *tr* der rechte Teilbaum von w. Im Induktionsschritt kann nun verwendet werden, dass $h > 1$ gilt und w ein innerer Knoten sein muss. *tl* und *tr* haben Schwarz-Höhe $bh(w)-1$ wenn ihre Wurzel schwarz ist und Schwarz-Höhe $bh(w)$ wenn ihre Wurzel rot ist. Ihre Höhe ist $h-1$ und somit enthalten sie nach Induktionsnahme mindestens $2^{bh(w)-1}-1$ innere Knoten. Aufaddieren ergibt die Behauptung.

$$\begin{aligned} & m \geq 2^{bh(w)-1} - 1 + 1 + 2^{bh(w)-1} - 1 = 2^{bh(w)} - 1 \\ \Rightarrow & log_2(m+1) \geq bh(w) \end{aligned}$$

Es gilt folgender Zusammenhang, da höchstens jeder zweite Knoten in einem Pfad rot sein kann

$$\begin{aligned} & h(w) \leq 2 \cdot bh(w) + 1 \\ \Rightarrow & \frac{h(w)-1}{2} \leq bh(w) \end{aligned}$$

Einsetzen liefert:

$$\begin{aligned} & log_2(m+1) \geq \frac{h(w)-1}{2} \\ \Rightarrow & 2 \cdot log_2(m+1) + 1 \geq h(w) \\ \Rightarrow & h(w) = O(\log m) \end{aligned}$$

Es kann nur maximal doppelt so viele Blätter wie innere Knoten geben. Daraus folgt.

$$\begin{aligned} & n \leq 3m \\ \Rightarrow & h(w) = O(\log n) \end{aligned}$$

□

suchen hat also Laufzeit $O(\log n)$. *einfügenFixup* mit übergebenen Knoten v Einfügen hat Kosten $O(\log n)$ für suchen und $O(\log n)$ für einfügenFixup, insgesamt also $O(\log n)$.

2.2 Tango-Baum konformes vereinigen

Hier wird die *vereinigen(RBT T_1, Schlüssel k, RBT T_2)* eingeführt wie es ein Tango-Baum von seiner Hilfsstruktur verlangt. Sei K_1 die Schlüsselmenge von T_1 und K_2 die von T_2. Die Operation gibt eine Referenz auf die

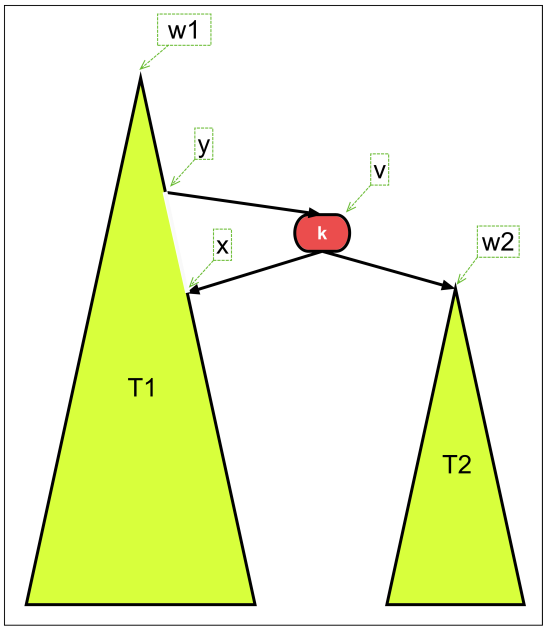

Abbildung 10: Beispielhaftes vereinigen zweier RBT unterschiedlicher Schwarz-Höhe, nach Schritt 1.

Wurzel eines vereinigten RBT T mit Schlüsselmenge $K_1 \cup K_2 \cup \{k\}$ zurück, dabei werden T_1 und T_2 zerstört. An die Parameter wird die Vorbedingung $(\forall i \in \mathbb{N} : i < k) \wedge (\forall j \in \mathbb{N} : k < j)$ gestellt.
Es werden im ersten Schritt der Ausführung drei Fälle unterschieden, wobei wieder der erste zutreffende Fall in aufsteigender Reihenfolge ausgewählt wird.
Fall 1: $bh(T_1) = bh(T_2) = 0$
In diesem Fall wird ein roter Knoten v mit Schwarz-Höhe 1 erzeugt. An diesem werden zwei Sonderknoten angefügt. v ist die Wurzel von T.
In den restlichen Fällen ist nun zumindest ein Baum vorhanden, der wenigstens einen inneren Knoten beinhaltet und somit über eine Wurzel verfügt. Der RBT mit der geringeren Schwarz-Höhe wird dabei an den mit der höheren „seitlich angefügt“. Abbildung 10 zeigt dies beispielhaft. Sind die Schwarz-Höhen gleich wird wie in Abbildung 11 vorgegangen. Nun werden die verbleibenden Fälle beschrieben.
Fall 2: $bh(T_2) \leq bh(T_1)$
In diesem Fall wird T_2 bei T_1, mit Hilfe von k, so angefügt, dass die Schwarz-Höhe jedes Knoten unverändert bleibt. Es sei w_1 die Wurzel von T_1. Es sei p ein Pfad $(r_0, r_1, ..., r_l)$ in T_1, so dass $r_0 = w_1$, r_l ein Blatt ist und $\forall i \in \{1, 2, ...l\}$: r_i ist rechtes Kind von r_{i-1}. p ist also der am weitesten

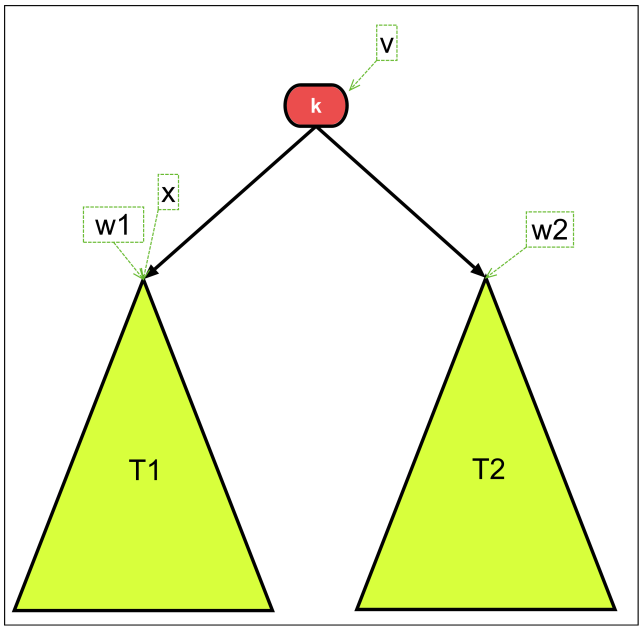

Abbildung 11: Beispielhaftes vereinigen zweier RBT gleicher Schwarz-Höhe, nach Schritt 1

rechts liegende Pfad von der Wurzel zu einem Blatt. Sei x der schwarze Knoten in p, mit $bh(x) = bh(w_2)$. x muss existieren denn $bh(w_1) \geq bh(w_2)$ und $bh(r_l) \leq bh(w_2)$, außerdem sind w_1 und r_l schwarz.
Nun wird ein neuer roter Knoten v mit Schlüssel k und Schwarz-Höhe $bh(x)+1$ erzeugt. Als linkes Kind von v wird x gesetzt, als rechtes Kind w_2. Ist x die Wurzel in T_1, so ist v die Wurzel von T. Ansonsten ist x rechtes Kind eines Knoten y und das rechte Kind von y wird auf v gesetzt. Außerdem ist dann w_1 die Wurzel von T.
Fall 3: $bh(T_1) < bh(T_2)$
Dieser Fall ist fast symmetrisch zu Fall 2, jedoch kann der neue Knoten nicht zur Wurzel von T werden, da $bh(T_1) \neq bh(T_2)$. Es wird T_1 bei T_2, mit Hilfe von k angefügt. Es sei w_2 die Wurzel von T_2. Es sei p ein Pfad $(r_0, r_1, ..., r_l)$ in T_2, so dass $r_0 = w_2$, r_l ein Blatt ist und $\forall i \in \{1, 2, ...l\}: r_i$ ist linkes Kind von r_{i-1}. p ist also der am weitesten links liegende Pfad von der Wurzel zu einem Blatt. Sei x der schwarze Knoten in p, mit $bh(x) = bh(w_1)$. x muss existieren denn $bh(w_2) < bh(w_1)$ und $bh(r_l) \leq bh(w_1)$, außerdem sind w_2 und r_l schwarz.
Nun wird ein neuer roter Knoten v mit Schlüssel k und Schwarz-Höhe $bh(x)+1$ erzeugt. Als rechtes Kind von v wird x gesetzt, als linkes Kind w_2. x ist linkes Kind eines Knoten y und das linke Kind von y wird auf v gesetzt. w_2

ist die Wurzel von T.

Resultat nach der Fallbehandlung Dass ein Baum mit Schlüsselmenge $K_1 \cup K_2 \cup \{k\}$ entstanden ist, erkennt man direkt an den Abbildungen 10 und 11. Aufgrund der Vorbedingung an die Parameter, muss T auch ein BST sein. Es müssen aber wieder die fünf Eigenschaften eines RBT betrachtet werden:

1. Es ist immer noch jeder Knoten entweder rot oder schwarz.
2. Gilt $bh(T_1) \neq bh(T_2)$ so wurde mit w_1 oder w_2 ein schwarzer Knoten zur Wurzel von T. Anderenfalls ist v die rote Wurzel von T und diese Eigenschaft ist verletzt.
3. Aufgrund der Sonderknoten sind die Blätter immer noch schwarz.
4. Da T_1 und T_2 RBTs waren muss nur die Situation um v betrachtet werden. v hat in jedem Fall schwarze Kinder. Gilt $bh(T_1) \neq bh(T_2)$ könnte v jedoch einen roten Vater y haben.
5. Die Schwarz-Höhe von v ist korrekt gesetzt. Existiert y so hat sich seine Schwarz-Höhe nicht verändert, denn v ist rot. Bei keinem anderen Knoten hat sich bezüglich bzgl. der Schwarz-Höhe etwas geändert.

Wir sind also in der Situation dass entweder Eigenschaft zwei verletzt ist oder Eigenschaft vier. Wenn Eigenschaft vier verletzt ist dann nur an Knoten v. Das ist genau die Situation für die einfügenFixup entworfen wurde.
In Schritt zwei wird also einfügenFixup mit Parameter v aufgerufen und die Wurzel des resultierenden RBT zurückgegeben.

Laufzeit Sei n_1 die Anzahl der Knoten von T_1, sei n_2 die Anzahl der Knoten von T_2 und $n = n_1 + n_2$. Der Tango-Baum fordert von seiner Hilfsstruktur eine Laufzeit von $O(\log n)$ für die eben vorgestellte Operation. Das ablaufen eines Pfades in T_1 oder T_2 zum Finden von x liegt in $O(\log(n))$. v erzeugen und in die Struktur einzubinden benötigt konstante Zeit und einfügenFixup benötigt $O(\log(n))$ Zeit.
$O(\log(n)) + O(1) + O(\log(n)) = O(\log(n))$
Die Vorgabe des Tango-Baumes wird also eingehalten.
Für das nächste Kapitel wird noch eine genauere Betrachtung der Laufzeit benötigt. Es sei $d = |bh(T_1) - bh(T_2)|$. Die Suche nach x endet spätestens nach dem ein Pfad der Länge $2d + 1$ betrachtet wurde. Dabei steht die 1 für x selbst. Zu jedem schwarzen Knoten über x könnte noch ein roter kommen.

einfügenFixup wird mit Parameter v aufgerufen. Sei w die Wurzel von T. v ist ein roter Knoten mit $bh(v) = bh(w) - d + 1$ und liegt in Ebene $2d + 2$ oder höher. Deshalb führt *einfügenFixup* maximal $d + 1$ Iterationen durch.

2.3 Tango-Baum konformes aufteilen

Auch *aufteilen (RBT T, Schlüssel k)* wird so vorgestellt, wie es als Hilfsstruktur für einen Tango-Baum benötigt wird. Vorbedingung an die Parameter ist, dass k in der Schlüsselmenge K von T vorhanden ist. Zurückgegeben wird eine Referenz auf den Knoten v mit Schlüssel k. Linkes Kind von v ist die Wurzel eines RBT T_L mit Schlüsselmenge K_L, wobei gilt $K_L = \{i \mid i \in K \wedge i < k\}$. Rechtes Kind von v ist die Wurzel eines RBT T_2 mit Schlüsselmenge K_R, wobei gilt $K_R = \{i \mid i \in K \wedge i > k\}$. *aufteilen* gibt also in den meisten Fällen keinen RBT zurück. Die Operation setzt zunächst T_L auf den linken Teilbaum von v und T_R auf den rechten Teilbaum von v. Eventuell müssen die Wurzeln schwarz gefärbt werden. Dann wird sich auf dem Pfad von v zur Wurzel von T Knoten für Knoten nach oben gearbeitet. Ist der Schlüssel eines Knotens kleiner als k, so wird dieser Schlüssel und der linke Teilbaum des Knotens zu T_L hinzugefügt. Dies übernimmt unsere *vereinigen* Operation. Ist der Schlüssel größer als k, so wird dieser Schlüssel und der rechte Teilbaum des Knotens zu T_R hinzugefügt. Folgende Aufzählung beschreibt den Vorgang genauer. Es sei $(v_0, v_1, ..., v_m)$ der Pfad von $v = v_0$ zur Wurzel v_m.

1. Verwende die *suchen* Operation um den Knoten v_0 mit Schlüssel k zu finden.
2. Setze den linken Teilbaum von v_0 als T_L, den rechten als T_R. Löse beide Teilbäume aus T heraus.
3. Färbe die Wurzeln von T_L und T_R schwarz.
4. $\forall i \in \{1, 2, .., m\}$ aufsteigend sortiert. Ist der Schlüssel k_i von v_i kleiner als k, vereinige T_L mit dem aus T herausgelösten linken Teilbaum von v_i, mit k_i als dritten Parameter. Ansonsten vereinige T_R mit dem aus T herausgelösten rechten Teilbaum von v_i, mit k_i als dritten Parameter.
5. Füge T_R rechts an v an, T_L links.
6. Gib v zurück.

Das T_L und T_R die gewünschten RBTs sind wird leichter erkannt, wenn man sich den Pfad von oben nach unten, also $(v_m, .., v_0)$, betrachtet. v_m und einer der beiden Teilbäume wird korrekt zugeordnet. Die Wurzel des anderen

Teilbaumes von v_m ist v_{m-1}. Alle Schlüssel die nicht im Teilbaum mit Wurzel v_{m-1} liegen sind somit korrekt zugeordnet. Dieses vorgehen iteriert bis man auf v trifft. Die Schlüssel im Teilbaum mit Wurzel v werden korrekt zugeordnet.

Laufzeit Der Tango-Baum fordert eine Laufzeit von $O(\log(n))$ von seiner Hilfsstruktur für *aufteilen*. Punkt 1 kostet $O(\log(n))$ Zeit. Durchführen von Punkt 2,3,5 und 6 kostet $O(1)$. Punkt 4 führt $O(\log(n))$ Aufrufe von *vereinigen* durch. Das ergibt $O(\log(n)\log(n))$, was dann auch eine obere Schranke für die Gesamtlaufzeit ist. Diese Schranke ist für unseren Einsatzzweck jedoch zu hoch.

3 dd

3.1 Einführung der amortisierten Laufzeitanalyse

Sei $i \in \{0, .., m\}$. Bei der **amortisierten Laufzeitanalyse** wird eine Folge von m Operationen betrachtet. Hierbei kann es sich m mal um die gleiche Operation handeln, oder auch um verschiedene. Die **tatsächlichen Kosten** t_i stehen für die exakten Kosten zum ausführen der i-ten Operation. Durch aufaddieren der tatsächlichen Kosten jeder einzelnen Operation erhält man **tatsächlichen Gesamtkosten**. Stehen für die Laufzeit der Operationen jeweils nur obere Schraken zur Verfügung, kann man mit diesen genau so vorgehen, um eine obere Schranke für die Gesamtlaufzeit zu erhalten. So erzeugte obere Schranken können jedoch unnötig hoch sein. Die Idee bei einer amortisierten Analyse ist es, bereits eingesparte Zeit durch schnell ausgeführte Operationen, den noch folgenden Operationen zum Verbrauchen zur Verfügung zu stellen. Dabei wird insbesondere der aktuelle Zustand der zugrunde liegenden Datenstruktur vor und nach einer Operation betrachtet. Hier soll die amortisierte Laufzeitanalyse verwendet werden um im folgenden Abschnitt eine niedrigere obere Schranke als $O(\log(n))$ für einfügenFixup zu finden. Es gibt drei Methoden zur amortisierten Analyse, hier wird die **Potentialfunktionmethode** verwendet.

Potentialfunktionmethode Eine Potentialfunktion $\Phi(D)$ ordnet einem Zustand einer Datenstruktur D eine natürliche Zahl, **Potential** genannt, zu. Es bezeichnet $\Phi(D)_i$ das Potential von D nach Ausführung der i-ten Operation. t_i steht für die **tatsächlichen Kosten** zum durchführen der i-ten Operation. Dabei handelt es sich um die exakten Kosten die beim Die

amortisierten Kosten a_i einer Operation berücksichtigen die von der Operation verursachte Veränderung am Potential, $a_i = t_i + \Phi(D)_i - \Phi(D)_{i-1}$. Um die **amortisierten Gesamtkosten** A zu berechnen bildet man die Summe der amortisierten Kosten aller Operationen.

$$A = \sum_{i=1}^{m} a_i = \sum_{i=1}^{m} \left(t_i + \Phi\left(D\right)_i - \Phi\left(D\right)_{i-1}\right) = \Phi\left(D\right)_m - \Phi\left(D\right)_0 + \sum_{i=1}^{m} t_i$$

Folgendes gilt für die Summe der t_i:

$$\sum_{i=1}^{m} t_i = \sum_{i=1}^{m} \left(a_i - \Phi\left(D\right)_i + \Phi\left(D\right)_{i-1}\right) = \Phi\left(D\right)_0 - \Phi\left(D\right)_m + \sum_{i=1}^{m} a_i$$

$$\Rightarrow \left(\Phi\left(D\right)_m \geq \Phi\left(D\right)_0 \Rightarrow \sum_{i=1}^{m} a_i \geq \sum_{i=1}^{m} t_i\right)$$

Ist das Potenzial nach Ausführung der Operationsfolge also nicht kleiner als zum Beginn, dann sind die amortisierten Gesamtkosten eine obere Schranke für die tatsächlichen Gesamtkosten. Die wesentliche Aufgabe ist es nun eine Potentialfunktion zu finden, bei der die amortisierten Gesamtkosten möglichst niedrig sind und für die gilt $\Phi\left(D\right)_m \geq \Phi\left(D\right)_0$. Dies wird jetzt noch an einem einfachen Beispiel demonstriert, bevor einfügenFixup betrachtet wird.

Potentialfunktionmethode am Beispiel eines Stack Der Stack verfügt wie gewöhnlich über eine Operation *push* zum ablegen eines Elementes auf dem Stack und über *pop* zum entfernen des oben liegenden Elementes. Zusätzlich gibt es eine Operation *popAll*, die so oft *pop* aufruft, bis der Stack leer ist. Sei n die Anzahl der Elemente die maximal im Stack enthalten sein kann. *push* und *pop* können in konstanter Zeit durchgeführt werden und wir berechnen jeweils eine Kosteneinheit. Für die Laufzeit von *popAll* gilt $O(n)$, da *pop* bis zu n mal aufgerufen wird. Für die Gesamtlaufzeit einer Folge von m Operationen kann sicher $O(mn)$ angegeben werden. Mit einer amortisierten Analyse wird nun aber $O(m)$ für *popAll* gezeigt. Als Φ verwenden wir eine Funktion, welche die aktuelle Anzahl der im Stack enthaltenen Elemente zurück gibt. Φ_0 setzen wir auf 0, dass heißt wir starten mit einem leeren Stack. *push* erhöht also das Potential um eins, während *pop* es um eins vermindert. Nun werden die amortisierten Kosten bestimmt.

$$\begin{aligned} a_{push} &= t_{push} + \Phi i - \Phi i - 1 &&= 2 \\ a_{pop} &= t_{pop} + \Phi i - \Phi i - 1 &&= 0 \\ a_{popAll} &= n \cdot a_{pop} &&= 0 \end{aligned}$$

Alle drei Operationen haben konstante amortisierte Kosten. Auf jedem Fall gilt $\Phi_m \geq \Phi_0 = 0$ Damit gilt für die Ausführungszeit der Folge $O(m)$.
Bei diesem einfachen Beispiel ist sofort klar warum es funktioniert. Aus einem zu Beginn leerem Stack kann nur entfernt werden, was zuvor eingefügt wurde. *push* zahlt für die Operation, welche das eingefügte Element eventuell wieder entfernt gleich mit, bleibt bei den Kosten aber konstant. Deshalb kann *pop* amortisiert kostenlos durchgeführt werden, wodurch einer der beiden Faktoren zur Berechnung der Kosten von *popAll* zu 0 wird.

Literatur

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.