

Bachelorarbeit

Andreas Windorfer

3. Mai 2020

Zusammenfassung

Inhaltsverzeichnis

1 Fazit	4
2 Binäre Suchbäume	4
2.1 Definition binärer Suchbaum	4
2.2 Weitere Begriffe und Eigenschaften zum binären Suchbaum . .	5
2.3 Rot-Schwarz-Baum	14
2.3.1 Suchen im Rot-Schwarz-Baum	15
2.3.2 Einfügen in den Rot-Schwarz-Baum	16
2.3.3 Löschen aus dem Rot-Schwarz-Baum	21
2.3.4 Laufzeit der Grundoperationen	22

1 Fazit

2 Binäre Suchbäume

Es gibt viele Varianten von binären Suchbäumen mit unterschiedlichen Anforderungen und Leistungsdaten. In diesem Kapitel werden binäre Suchbäume im Allgemeinen beschrieben. Außerdem werden Begriffe definiert, die in den nachfolgenden Kapiteln verwendet werden.

2.1 Definition binärer Suchbaum

Ein **Baum** T ist ein zusammenhängender, gerichteter Graph, der keine Zyklen enthält. In einem nicht leerem Baum gibt es genau einen Knoten ohne eingehende Kante, diesen bezeichnet man als **Wurzel**. Alle anderen Knoten haben genau eine eingehende Kante. Jeder Knoten v in T ist Wurzel eines **Teilbaumes** $T(v)$, der v und alle von v erreichbaren Knoten enthält. Knoten ohne ausgehende Kante nennt man **Blatt**, alle anderen Knoten werden als **innere Knoten** bezeichnet. Enthält der Baum eine Kante von Knoten v_1 zu Knoten v_2 so nennt man v_2 ein **Kind** von v_1 und v_1 bezeichnet man als den **Vater** von v_2 . Die Wurzel hat also keinen Vater, alle anderen Knoten genau einen.

Bei einem **binären Baum** kommt folgende Einschränkung hinzu:

Ein Knoten hat maximal zwei Kinder.

Entsprechend ihrer Zeichnung benennt man die Kinder in Binärbäumen als **linkes Kind** oder **rechtes Kind**. Sei w das linke bzw. rechte Kind von v , dann bezeichnet man den Teilbaum mit Wurzel w als **linken Teilbaum** bzw. **rechten Teilbaum** von v .

Bei einem **binären Suchbaum** ist jedem Knoten ein innerhalb der Baumstruktur ein eindeutiger **Schlüssel** aus einem **Universum** zugeordnet. Als Universum kann jede Menge M verwendet werden, auf der eine totale Ordnung definiert ist. Auf totale Ordnungen wird in der dieser Einleitung noch eingegangen. Hier und den folgenden Kapiteln wird als Universum immer \mathbb{N} verwendet. Die in einem binären Suchbaum enthaltenen Schlüssel bezeichnen wir als seine **Schlüsselmenge**. Damit aus dem binären Baum ein binärer Suchbaum wird, benötigt man noch folgende Eigenschaft:

Für jeden Knoten im binären Suchbaum muss gelten, dass alle Schlüssel die in seinem linken Teilbaum enthaltenen sind kleiner sind als der eigene Schlüssel und alle im rechten Teilbaum enthaltenen größer.

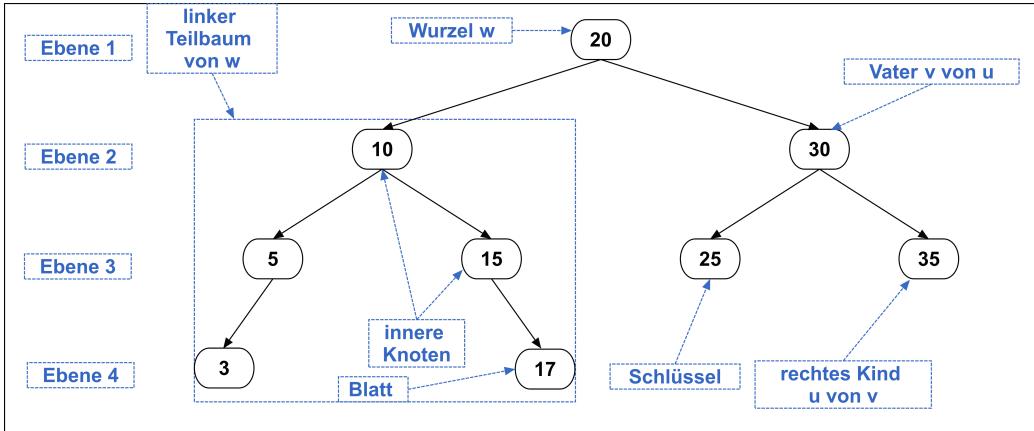


Abbildung 1: Ein binärer Suchbaum

Es gibt eine rekursive Definition für binäre Suchbäume, aus der die gerade geforderten Eigenschaften direkt ersichtlich sind. Diese soll auch hier verwendet werden.

Definition 2.1. Binärer Suchbaum

1. Der leere Baum ohne Knoten ist ein binärer Suchbaum.
2. Der Baum mit dem einzigen Knoten v der Schlüssel k_v enthält ist ein binärer Suchbaum.
3. Es seien T_1 und T_2 binäre Suchbäume mit Schlüsselmenge K_1 bzw. K_2 . Sei $i \in \mathbb{N}$, mit $\max(K_1) < i < \min(K_2)$. Erzeuge einen neuen Knoten w mit Schlüssel i . Setze T_1 als linken Teilbaum von w und T_2 als rechten Teilbaum von w . Die so entstandenen Struktur ist ein binärer Suchbaum mit Wurzel w .
4. Eine Struktur die sich nicht durch Anwenden von Punkt 1, 2 und 3 erzeugen lässt, ist kein binärer Suchbaum.

Anstatt binärer Suchbaum schreibt man häufig **BST** für Binary Search Tree. Diese Abkürzung wird hier ab jetzt auch verwendet.

2.2 Weitere Begriffe und Eigenschaften zum binären Suchbaum

Zwei verschiedene Knoten mit dem selben Vater nennt man **Brüder**. Ein **Pfad** P_{jk} ist eine Folge von Knoten (v_0, v_1, \dots, v_n) , mit $v_0 = v_j$, $v_n = v_k$

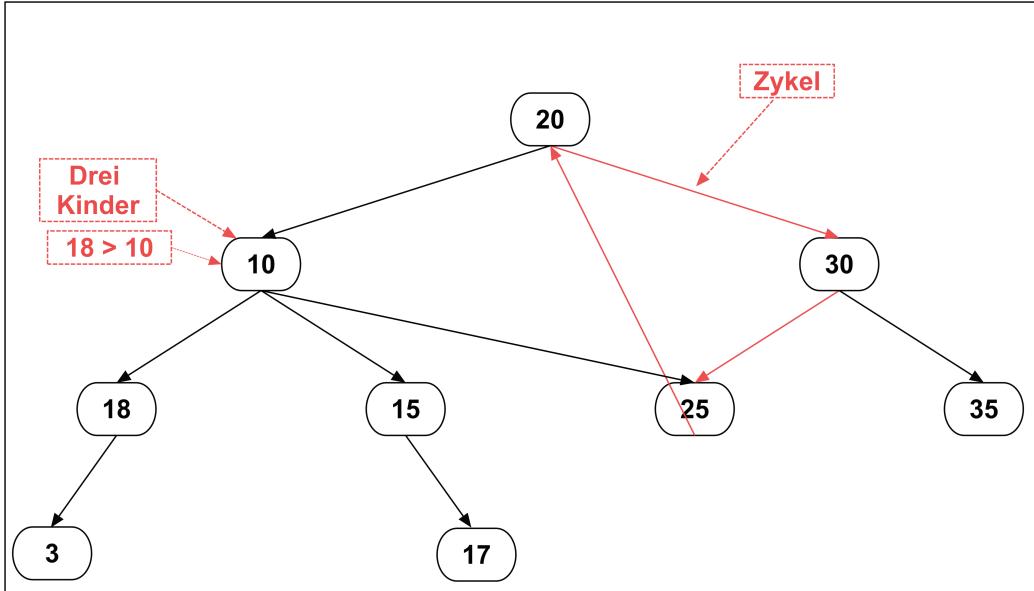


Abbildung 2: Kein binärer Suchbaum

und $\forall i \in \{1, 2, \dots, n\}$: v_{i-1} ist Vater von v_i . n ist die **Länge des Pfades**. Die Knoten v_0 bis v_{n-1} sind **Vorfahren** von v_n . Den Knoten in einem BST wird auch eine **Tiefe** und eine **Höhe** zugeteilt. Für einen Knoten v gilt, dass die Länge des Pfades von der Wurzel zu ihm seiner Tiefe entspricht. Sei l maximale Länge eines von v aus startenden Pfades. Die Höhe $h(v)$ von v ist dann $l+1$. Die Höhe der Wurzel entspricht der **Höhe des Gesamtbaumes**. Einen BST T mit Gesamthöhe h_T unterteilt man von oben nach unten in die **Ebenen** $1, 2, \dots, h_T$. Wobei die Wurzel in der Ebene eins liegt, deren Kinder in der Ebene zwei usw. Enthält eine Ebene ihre maximale Anzahl an Knoten nennt man sie **vollständig besetzt**.

Da im linken Teilbaum nur kleinere Schlüssel vorhanden sein dürfen und im rechten Teilbaum nur größere, kann man die Schlüsselmenge eines binären Suchbaumes, von links nach rechts, in aufsteigender Form ablesen. Denn angenommen es gibt zwei Knoten v_l mit Schlüssel k_l und v_r mit Schlüssel k_r , so dass $k_l > k_r$ gilt und v_l weiter links im Baum liegt als v_r . Ist ein v_l Vorfahre von v_r , so enthält der rechte Teilbaum von v_l einen Schlüssel der kleiner ist als k_l . Ist ein v_r Vorfahre von v_l , so enthält der linke Teilbaum von v_r einen Schlüssel der größer ist als k_r . Ist keiner der Knoten Vorfahre des anderen, muss es zumindest einen gemeinsamen Vorfahren geben, denn dann kann weder v_r noch v_l die Wurzel des BST sein. Sei v_v der gemeinsame Vorfahrens mit der größten Tiefe. Der linke Teilbaum von v_v enthält dann einen größeren Schlüssel, als der rechte Teilbaum dieses Knotens. In

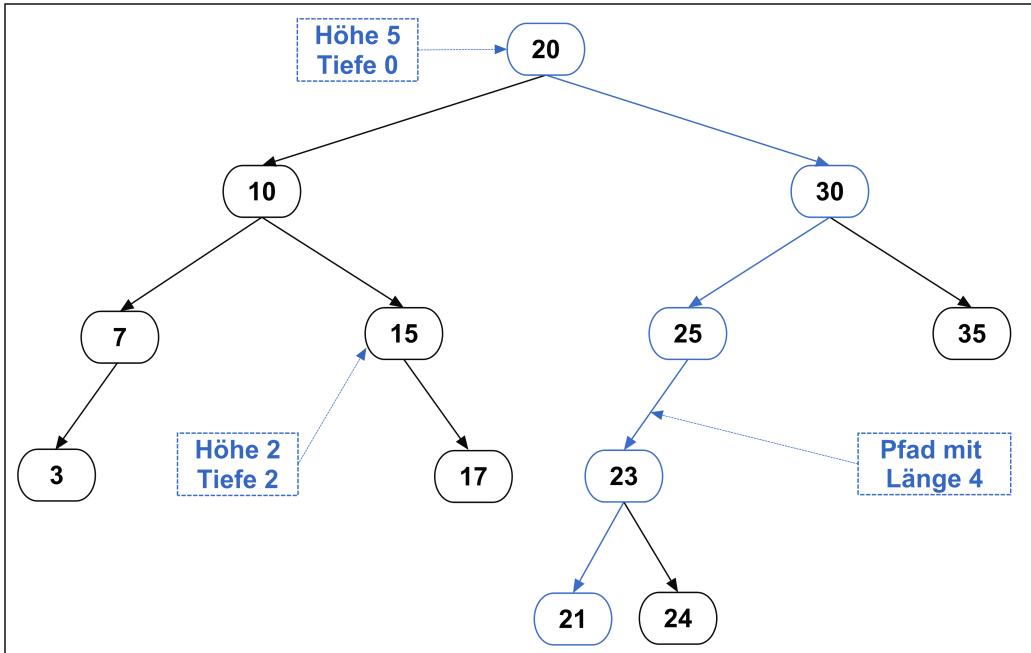


Abbildung 3: Ein weiterer binärer Suchbaum

jedem Fall erhält man einen Widerspruch zu der von BSTs geforderten Eigenschaft. Aus Platzgründen passiert es bei Zeichnungen von BSTs manchmal, dass ein Knoten in einem linken Teilbaum weiter rechts steht als die Wurzel des Teilbaumes, oder umgekehrt, weshalb man bei der Betrachtung solcher Zeichnungen etwas vorsichtig sein muss. Abbildung 4 enthält keine solche Konstellation.

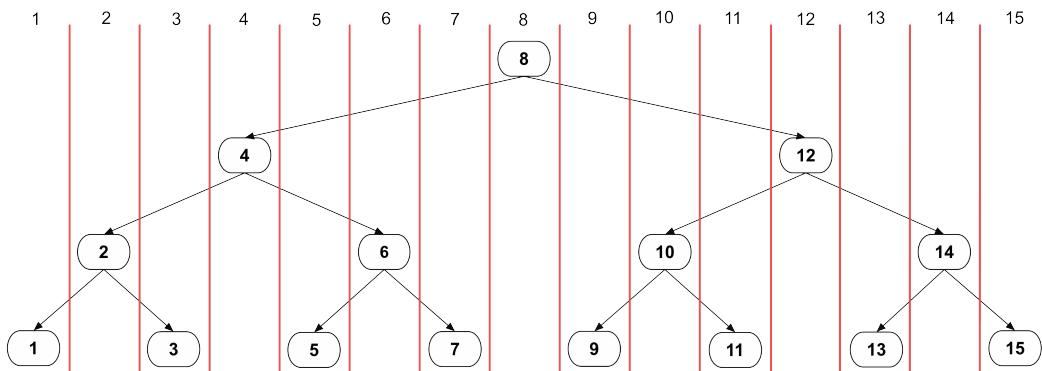


Abbildung 4: Schlüssel sind aufsteigend sortiert ablesbar.

Algorithmisch kann man sich die im BST enthaltenen Schlüssel aufsteigend sortiert durch eine **Inorder-Traversierung** ausgeben lassen. Es ist ein re-

kursives Verfahren, dass an der Wurzel startet und pro Aufruf drei Schritte ausführt.

Algorithmus *inorder* (**Knoten** v)

1. Existiert ein linkes Kind vl von v , rufe *inorder*(vl) auf.
2. Gib den Schlüssel von v aus.
3. Existiert ein rechtes Kind vr von v , rufe *inorder*(vr) auf.

Dass das Verfahren funktioniert sieht man leicht, durch Induktion über die Anzahl der Knoten n . Für $n = 0$ funktioniert es, da nichts ausgegeben wird. Wir nehmen nun an, dass die Ausgabe für BSTs mit Knotenzahl $\leq n$ korrekt ist. Sei T_1 ein BST mit Knotenzahl $n + 1$ und Wurzel w . Sowohl für den linken als auch für den rechten Teilbaum von w gilt, dass die Anzahl enthaltener Knoten $\leq n$ ist. Als erstes wird der linke Teilbaum von w korrekt ausgegeben, dann der Schlüssel von w selbst und zuletzt der rechte Teilbaum von w . Damit wurde auch für den Gesamtbaum die richtige Ausgabe erzeugt. Als **Vorgänger** eines Knoten v , mit Schlüssel k_v bezeichnet man den Knoten mit dem größten im BST enthaltenem Schlüssel k für den gilt $k < k_v$. Aus der Inorder-Traversierung kann man eine Anleitung zum Finden des Vorgängers ableiten. Wenn ein linker Teilbaum vorhanden ist, wird der größte Schlüssel in diesem direkt vor k ausgegeben. Andernfalls wird der Schlüssel des tiefsten Knotens, auf dem Pfad von der Wurzel zu v ausgegeben, bei dem v im rechten Teilbaum liegt. Als **Nachfolger** von v , bezeichnet man den Knoten mit dem kleinsten im BST enthaltenem Schlüssel k für den gilt $k > k_v$. Da dieser Schlüssel bei der Inorder-Traversierung direkt nach v ausgegeben wird, findet man den zugehörigen Knoten ganz links im rechten Teilbaum von v , falls ein solcher vorhanden ist. Ansonsten ist es der tiefste Knoten, auf dem Pfad von der Wurzel zu v , bei dem v im linken Teilbaum liegt. Abbildung 5 zeigt Vorgänger und Nachfolger eines Knotens.

Total geordnete Menge Eine Menge M wird als **total geordnet** bezeichnet wenn auf ihr eine zweistellige Relation \leq definiert ist, die folgende Eigenschaften erfüllt.

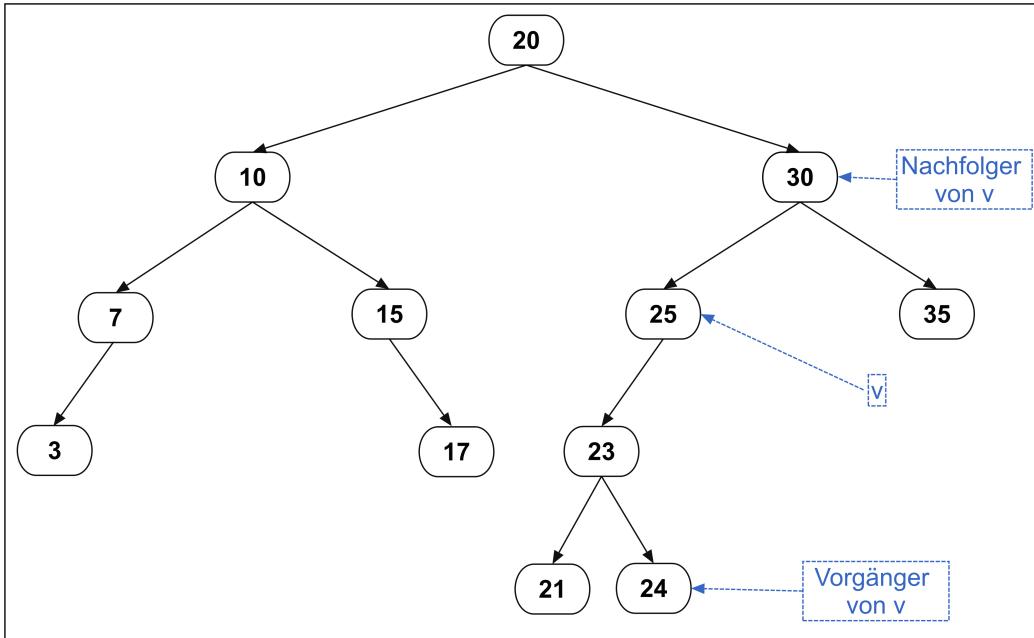


Abbildung 5: Darstellung von Vorgänger und Nachfolger.

Für alle $a,b,c \in M$ gilt:

1. $(a, a) \in R$ (reflexiv)
2. $(a, b) \in R \wedge (b, a) \in R \Rightarrow a = b$ (antisymmetrisch)
3. $(a, b) \in R \wedge (b, c) \in R \Rightarrow (a, c) \in R$ (transitiv)
4. $(a, b) \notin R \Rightarrow (b, a) \in R$ (total)

Die Eigenschaften 1,2 und 4 werden benötigt um für zwei beliebige Elemente aus der Menge feststellen zu können ob sie gleich sind. Oder bei Ungleichheit, welches Element weiter Links bzw. Rechts im BST liegen muss. Dafür wird z.B getestet ob die Elemente (a, b) und (b, a) in der Relation liegen. Eigenschaft 3 ist notwendig, denn liegt b weiter rechts im BST als a und c liegt weiter rechts als b , dann liegt c natürlich auch weiter rechts als a .

Die hier verwendete "Kleiner-Gleich-Beziehung auf den natürlichen Zahlen erfüllt alle Eigenschaften.

Verändern eines BST durch Rotationen. Wird ein BST T_1 durch eine Veränderung in einen BST T_2 überführt, kann es passieren dass sich Eigen-

schaften eines Knoten ändern. Um nicht immer erwähnen zu müssen auf welchen BST sich eine Aussage bezieht, wird es ab jetzt durchgängig so sein, dass sich ein Variablenname ohne angefügten Apostroph auf T_1 bezieht. Der gleiche Variablenname mit angefügtem Apostroph bezieht sich dann auf den Knoten in T_2 , mit dem gleichen Schlüssel. x bezieht sich beispielsweise auf einen Knoten in T_1 mit Schlüssel k . x' bezieht sich dann auf den Knoten mit Schlüssel k in T_2 .

Rotationen können verwendet werden um Änderungen an der Struktur eines BST durchzuführen, ohne eine der geforderten Eigenschaften zu verletzen. Es wird zwischen der **Linksrotation** und der **Rechtsrotation** unterschieden. Hier wird zunächst auf die in Abbildung 6 dargestellte. Linksrotation eingegangen. Sei x der Knoten auf dem eine Linksrotation durchgeführt wird. Sei z der Vater von x . z muss existieren, ansonsten darf auf x keine Rotation durchgeführt werden. Sei y der linke Teilbaum von x . Bei der Rotation nimmt x' den Platz von z ein. z' ist linkes Kind von x' . y' hängt rechts an z' . Für das Umhängen von y muss Platz sein, denn y' hängt da, wo x abgehängt wurde. Unabhängig von der Anzahl der im BST enthaltenen Knoten und der Ausführungsstelle im BST ist eine Linksrotation also mit dem Aufwand verbunden drei Zeiger umzusetzen. Zu beachten ist, dass die Höhe von x' und der Knoten in dessen, ansonsten unverändertem, rechtem Teilbaum jeweils um eins größer ist als die von x und den Knoten in dessen rechtem Teilbaum. Die Höhe Knoten im linken Teilbaum von z' sind jeweils um eins kleiner als die, im ansonsten unverändertem linken Teilbaum von z . Abbildung 7 zeigt die symmetrische Rechtsrotation. Man muss im obigen Beschreibung lediglich links durch rechts ersetzen und umgekehrt. Dass es durch eine Rotation zu keiner Verletzung der BST Eigenschaften kommt, sieht man den Abbildungen direkt an. In Abbildung 8 erkennt man, dass sich die Wirkung einer Rotation auf x durch eine gegenläufige Rotation auf z' aufheben lässt.

Grundoperationen Suchen, Einfügen und Löschen Hier geht es nur um die Standardvariante eines BST. Später werden Varianten gezeigt die von diesem Verhalten zum Teil deutlich abweichen. Es sei ein BST T gegeben. Die Operation **Suchen(Schlüssel k)** gibt den Knoten aus dem BST zurück, dessen Schlüssel mit k übereinstimmt. Die Operation startet bei der Wurzel und vergleicht den darin enthaltenen Schlüssel mit dem Gesuchten. Ist der gesuchte Schlüssel kleiner, muss er sich im linken Teilbaum des betrachteten Knoten befinden und die Suche wird bei dessen Wurzel fortgesetzt. Ist der Schlüssel größer, muss er sich im rechten Teilbaum befinden und die Suche wird bei dessen Wurzel fortgesetzt. Dieses Verhalten iteriert man solange bis

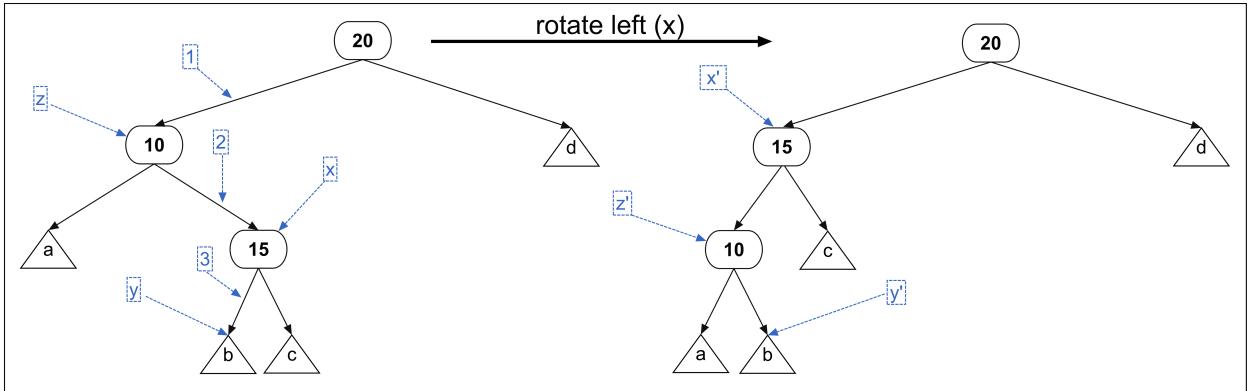


Abbildung 6: Linkssrotation auf Knoten x.

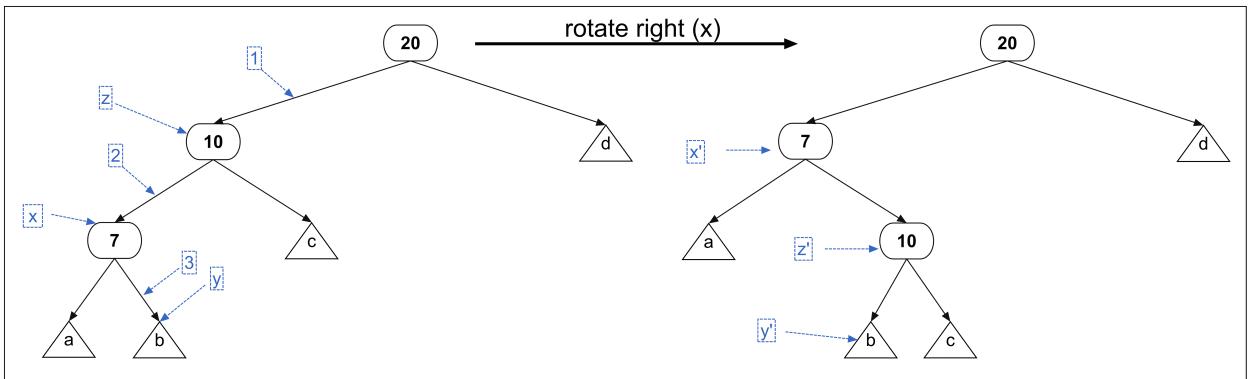


Abbildung 7: Rechtsrotation auf Knoten x.

der gesuchte Schlüssel gefunden ist, oder der Teilbaum bei dem die Suche fortgesetzt werden müsste, leer ist. Ist das Letztere der Fall, ist der gesuchte Schlüssel im Baum nicht vorhanden und es wird kein Knoten zurückgegeben. In keinem Fall kommt es zu einer Veränderung des BST.

Beim **Einfügen(Schlüssel k)** wird zunächst wie beim Suchen nach k verhalten. Findet man den Schlüssel wird das Einfügen abgebrochen und der BST bleibt unverändert. Wird ein leerer Teilbaum T_2 erreicht, wird ein neu erzeugter Knoten mit Schlüssel k an der Position von T_2 eingefügt.

Auch beim **Löschen(Schlüssel k)** wird sich zunächst wie beim Suchen verhalten. Ist k im BST nicht vorhanden wird abgebrochen und der BST bleibt unverändert. Ansonsten werden drei Fälle unterschieden. Sei v der Knoten mit Schlüssel k .

1. v ist ein Blatt:

v kann ohne weiteres aus dem BST entfernt werden.

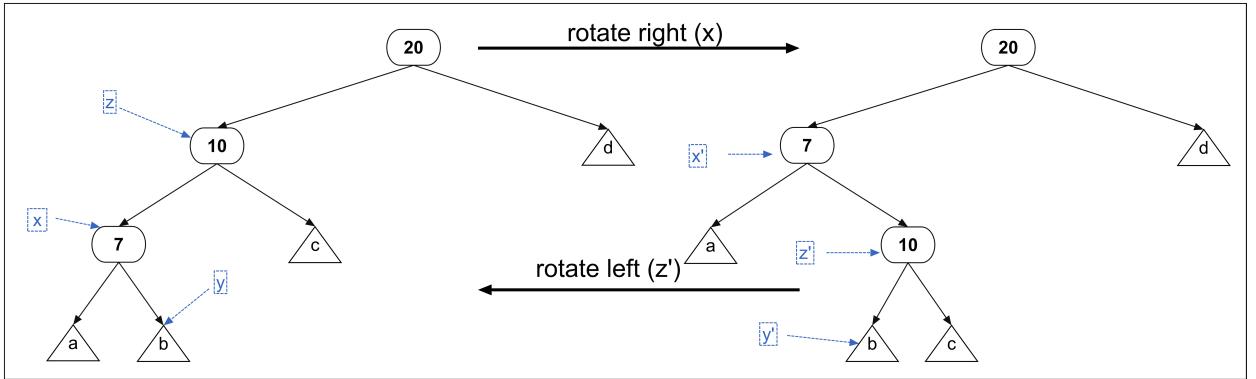


Abbildung 8: Gegenseitiges aufheben von Rotationen

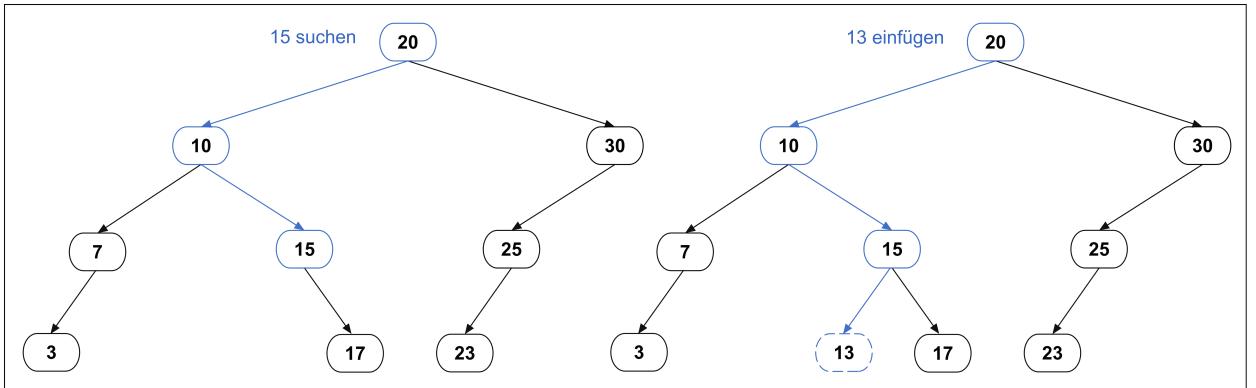


Abbildung 9: Links zeigt eine Suche nach dem Schlüssel 15. Rechts das Einfügen des Schlüssels 13

2. v hat genau ein Kind c :

Ist v die Wurzel kann er entfernt und c wird zur neuen Wurzel. Ansonsten ist v entweder ein linkes oder ein rechtes Kind eines Knoten w . c nimmt nun den Platz von v im BST ein. Das bedeutet, dass die Kante von w nach v entfernt wird. Außerdem wird eine Kante von w nach c so eingefügt, dass c wie zuvor v das linke bzw. rechte Kind von w wird.

3. v hat zwei Kinder:

Sei T_l der linke Teilbaum von v und T_r der Rechte. Sei z der Knoten mit dem kleinsten Schlüssel im rechten Teilbaum von v . Als Knoten mit dem kleinsten Schlüssel im rechten Teilbaum von v , kann w kein linkes Kind haben. Ist z ein Blatt wird es vom Baum abgehängt. Hat z ein rechtes Kind z_r , so nimmt dieses, analog zur Beschreibung im Fall 2, den Platz von z ein. Die ausgehende Kante von z wird noch entfernt, so dass z ein Knoten ohne verbliebene Kanten ist. z nimmt nun den

Platz von v ein, T_l wird links an z angefügt und T_r rechts. War v zu Beginn die Wurzel, so wird z zur neuen Wurzel.

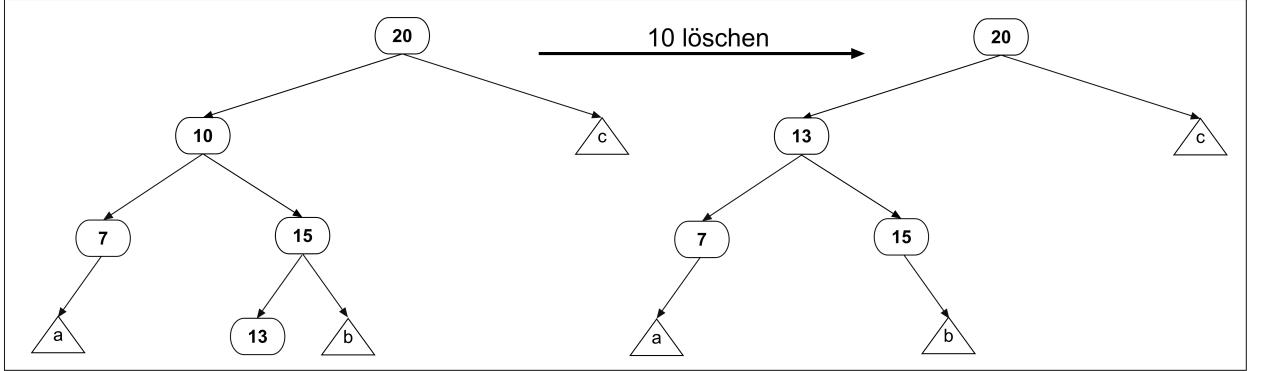


Abbildung 10: Löschen des Schlüssels 10

Die worst-case-Laufzeit der drei Operationen ist jeweils $O(h)$, wobei h die Höhe von T ist. Beim Suchen werden maximal h Knoten aus T betrachtet. Beim Einfügen überlagern die Kosten von Suchen, die konstanten Kosten für das Anhängen des neuen Knotens. Bei Löschen wird in Fall eins und zwei nach der Suche ebenfalls nur noch lokal beim gesuchten Knoten gearbeitet. Beim Löschen mit Fall drei muss man zunächst zum Knoten z gelangen, dafür sind maximal h Schritte notwendig. Danach muss man v erreichen, wozu ebenfalls maximal h Schritte notwendig sind. Die Kosten für das Entfernen und Hinzufügen von Kanten sind an beiden Stellen konstant.

Unterschiedliche Baumhöhen. Da die Höhe h eines BST T mit n Knoten entscheidend für die Laufzeit der vorgestellten Operationen ist, wird hier auf diese eingegangen. Die maximale Höhe n erreicht ein BST wenn es ein Blatt im BST gibt und jeder andere Knoten ein Halbblatt ist. Die Baumstruktur geht in diesem Fall über zu einer Listenstruktur über, dies wird als **entarten** bezeichnet. Minimal wird h wenn T **vollständig balanciert** ist. Das ist der Fall wenn alle Ebenen bis auf die Unterste vollständig besetzt sind.

Lemma 2.1. *Die Höhe eines vollständig balancierten BST T mit n Knoten ist $\lfloor \log_2(n) \rfloor + 1$.*

Beweis. Es sei $N(h)$ die maximale Anzahl an Knoten in einem vollständig balancierten BST mit Höhe h . $N(h)$ berechnet sich mit

$$N(h) = \sum_{i=0}^{h-1} 2^i = 2^h - 1$$

h ist minimal wenn gilt:

$$\begin{aligned} N(h-1) &< n \leq N(h) \\ \Leftrightarrow N(h-1) + 1 &\leq n < N(h) + 1 \end{aligned}$$

Einsetzen:

$$\begin{aligned} 2^{h-1} &\leq n < 2^h \\ \Rightarrow h &= \lfloor \log_2(n) \rfloor \end{aligned}$$

□

2.3 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. 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 *null* gesetzt wird, um sie eindeutig erkennen zu können. 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. Beide Kinder eines roten Knotens sind schwarz.

- 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, \dots, v_n) ein Pfad von einem Knoten v_0 zu einem Blatt v_n . Die Anzahl der schwarzen Knoten innerhalb (v_1, \dots, 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 eines Knoten x ist genau dann eindeutig wenn er Eigenschaft 5 nicht verletzt. Hält x Eigenschaft 5 ein, 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 $bh(x)$ wenn x rot ist. Im folgenden wird **RBT** (Red-Black-Tree) als Abkürzung für Rot-Schwarz-Baum verwendet.

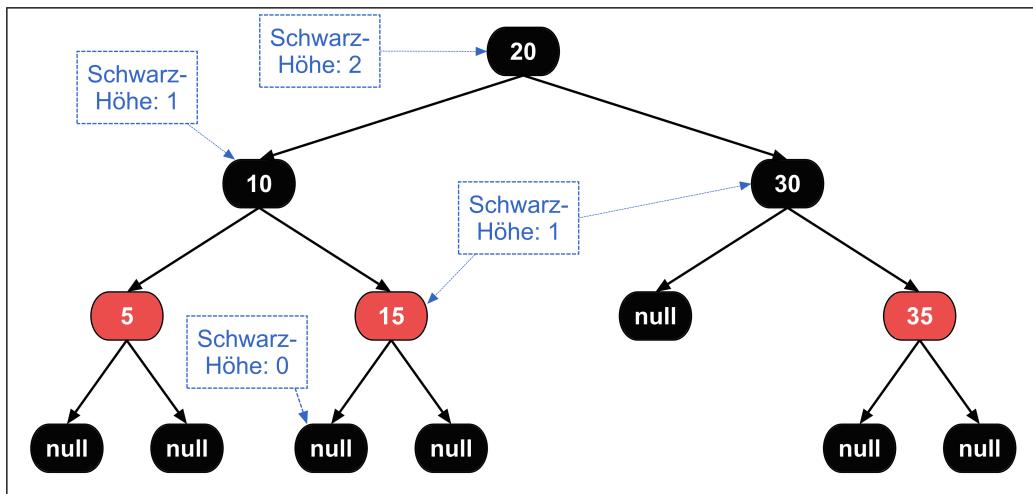


Abbildung 11: Rot-Schwarz-Baum ohne Verletzung von Eigenschaften.

2.3.1 Suchen im Rot-Schwarz-Baum

Die Suche unterscheidet sich nur in einem Punkt von der in 2.2 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 ohne einen Knoten zurückzugeben. Die Operation verändert den RBT nicht.

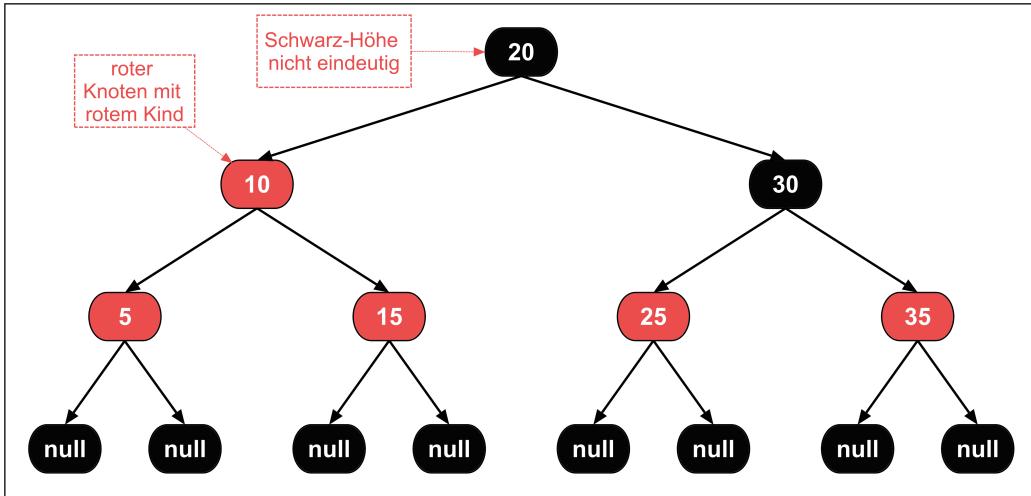


Abbildung 12: Rot-Schwarz-Baum bei dem Eigenschaft vier und fünf verletzt sind.

2.3.2 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 mit Schlüssel k 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 bleibt 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. Da der neue Knoten rot ist, ändern sich keine Schwarz-Höhen von bereits enthaltenen Knoten. Die Schwarz-Höhe des neuen Knoten ist immer 1. Eigenschaft fünf bleibt also erhalten.

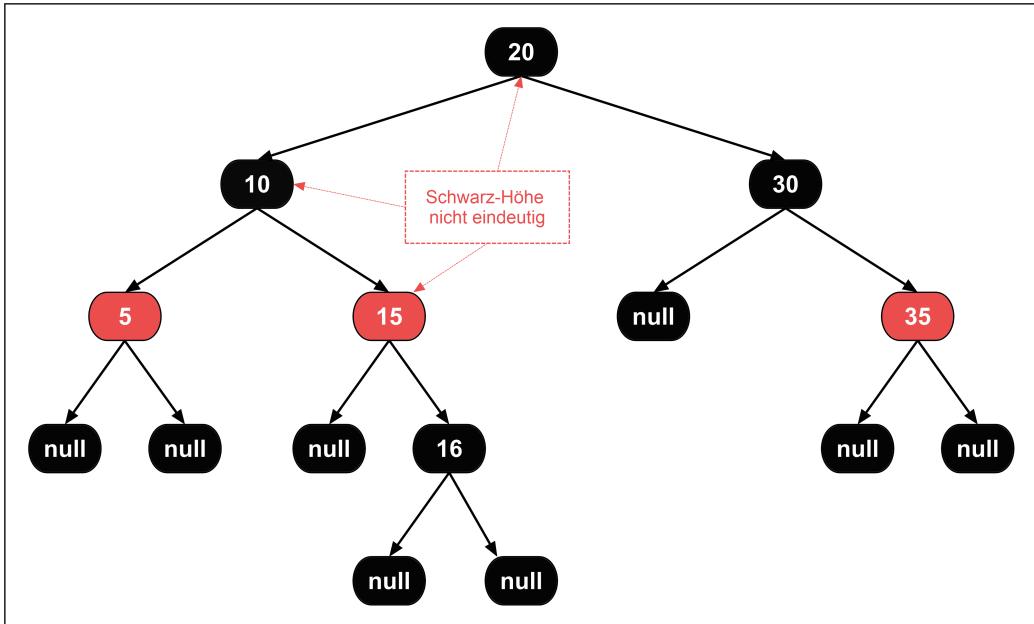


Abbildung 13: Rot-Schwarz-Baum bei dem Eigenschaft fünf verletzt ist.

Es können also die Eigenschaften zwei und vier betroffen sein. Jedoch nur eine von ihnen, denn Eigenschaft zwei wird genau dann verletzt wenn der neue Knoten der Einzige im Baum ist. Dann kann er aber keinen roten Vorgänger haben.

Zur Korrektur wird eine zusätzliche Operation, **einfügen-fixup** eingesetzt. Diese Operation arbeitet sich von der Einfügestelle solange nach oben in einer Schleife durch, bis alle Eigenschaften wieder erfüllt sind. Die Schleifenbedingung ist, dass eine Verletzung vorliegt. Dazu muss geprüft werden ob der betrachtete Knoten die rote Wurzel des Gesamtbaumes ist, oder ob er und sein Vater beide rot sind. Beim ersten Durchlauf wird der neu eingefügte Knoten übergeben. 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 einmal verletzt wird. 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 Laufzeit der Operation nur Knoten gibt, die entweder rot oder schwarz sind.

Fall 1: Die Wurzel ist rot: Dieser Fall wird behandelt in dem man die Wurzel schwarz färbt. Man muss noch zeigen, dass es durch das Umfärbeln zu keiner anderen Verletzung gekommen ist.

Betrachtung der Eigenschaften:

1. -
2. Die Wurzel wurde schwarz gefärbt.
3. Die Blätter sind unverändert.
4. Es wurden weder rote Knoten hinzugefügt, noch wurden Zeiger umgesetzt.
5. Das Umfärbeln 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 zweier aufeinanderfolgender roter Knoten. Der untere rote Knoten wird als x bezeichnet, der obere als y . Da Eigenschaft fünf nach jeder Iteration erfüllt ist muss y einen Bruder haben. Denn da y rot ist und Fall 1 nicht ausgewählt wurde, kann es nicht die Wurzel sein. Also muss auch y einen Vorgänger z haben. Da z kein Blatt(Sonderknoten) ist, müssen beide Kinder vorhanden sein.

Fall 2: y hat einen roten Bruder: Diesen Fall veranschaulicht Abbildung 14. 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. Somit ist der Vater von x nun einen 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 umgefärzte 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.

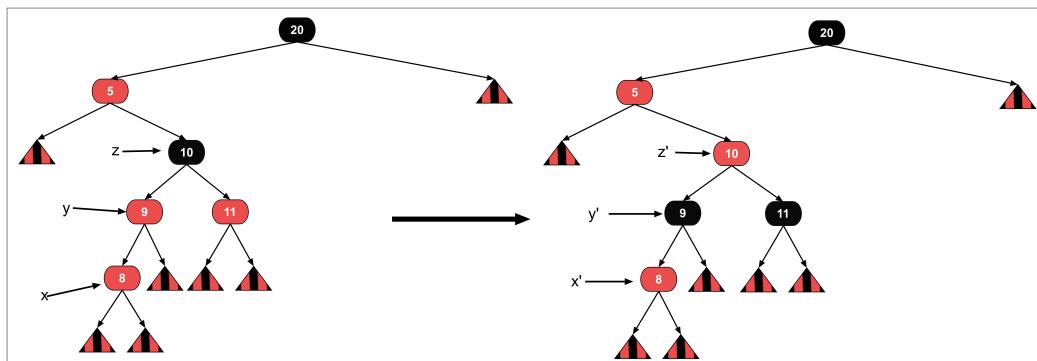


Abbildung 14: einfügen-fixup. Dargestellt ist Fall 2

Fall 3: y ist ein linkes Kind. x ist ein linkes Kind:

Abbildung 15 zeigt eine entsprechende Situation. Da in dieser Situation, die Wurzel schwarz sein muss, gibt es den Vater z von y . Da es nur eine Stelle im Baum geben kann an der Eigenschaft vier verletzt wird, muss z schwarz sein. Es wird nun 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 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 Stellen 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 von den Knoten 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 $bh(y') = mathit{bh}(x') = bh(z)$. 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: y ist ein linkes Kind. x ist ein rechtes Kind:

Dieser in Abbildung 16 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 einer Stelle 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: y ist ein rechtes Kind. x ist rechtes Kind:

Links/Rechts-Symmetrische Ausgangssituation zu Fall 3.

Fall 6: y ist ein rechtes Kind. x ist linkes Kind:

Links/Rechts-Symmetrische Ausgangssituation zu Fall 4.

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.

2.3.3 Löschen aus dem Rot-Schwarz-Baum

Evtl. noch machen, für tango eigentlich nicht notwendig

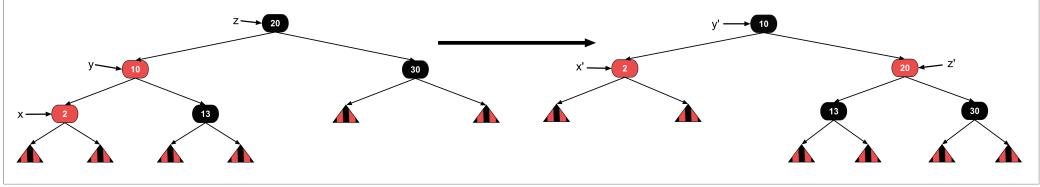


Abbildung 15: einfügen-fixup. Dargestellt ist Fall 3

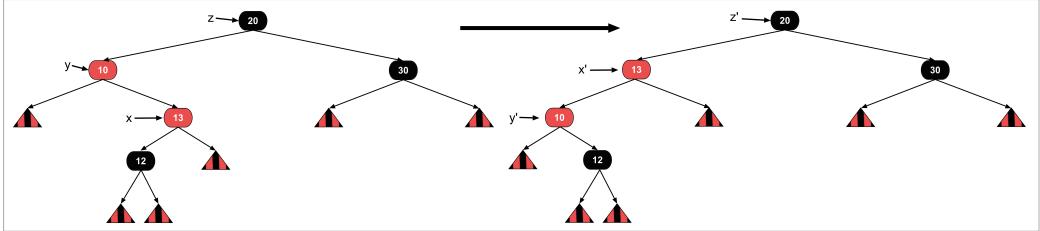


Abbildung 16: einfügen-fixup. Dargestellt ist Fall 4

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

Beweis. Sei w die Wurzel von T und m Zunächst wird gezeigt, dass T mindestens $2^{bh(w)} - 1$ innere Knoten enthält. Dies geschieht mit Induktion über h . Für $h = 0$ mit $2^0 - 1 = 0$ stimmt die Behauptung, denn der Baum ist leer. Für $h = 1$ mit $2^1 - 1 = 0$ stimmt die Behauptung ebenfalls, denn der Baum besteht aus einem einzigen Blatt. 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. Aufzählen ergibt die Behauptung.

$$m \geq 2^{bh(w)-1} - 1 + 1 + 2^{bh(w)-1} - 1 = 2^{bh(w)} - 1$$

Für einen Knoten x gilt folgender Zusammenhang, da höchstens jeder zweite Knoten in einem Pfad rot sein kann

$$h(x) \leq 2 \cdot bh(x) + 1$$

Es gilt also:

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

Damit gilt $h = O(\log m)$. Mit Blättern steigt die Höhe des Baumes um eins, damit ist auch das Lemma gezeigt.

□

Literatur