Bachelorarbeit

Andreas Windorfer

2. Mai 2020

Zusammenfassung

Inhaltsverzeichnis

1 Fazit

2 Binäre Suchbäume

Es gibt viele Varianten von binären Suchbäumen mit unterschiedlichen Anforderungen und Leistungsdaten. In diesem Kapitel werden binäre Suchbäume im Allgemeinem 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 **binärem 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 binärem 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.

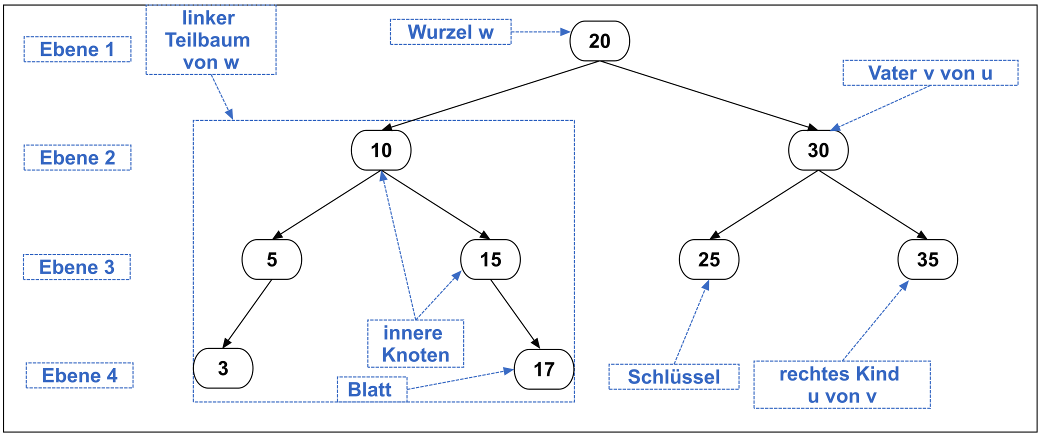

Abbildung 1: Ein binärer Suchbaum

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 $(v_0, v_1, \ldots, v_n)$, mit $v_0 = v_j$, $v_n = v_k$

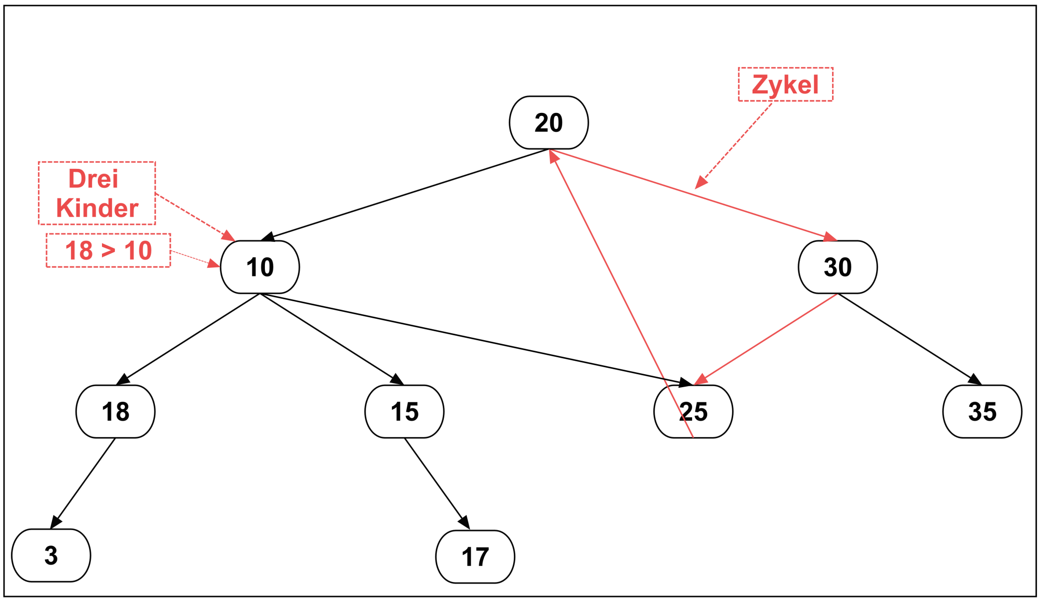

Abbildung 2: Kein binärer Suchbaum

und $\forall i \in \{1, 2, ..., n\}$: 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 von v ist dann $h+1$. 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...,* 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 aufsteigend sortierter 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

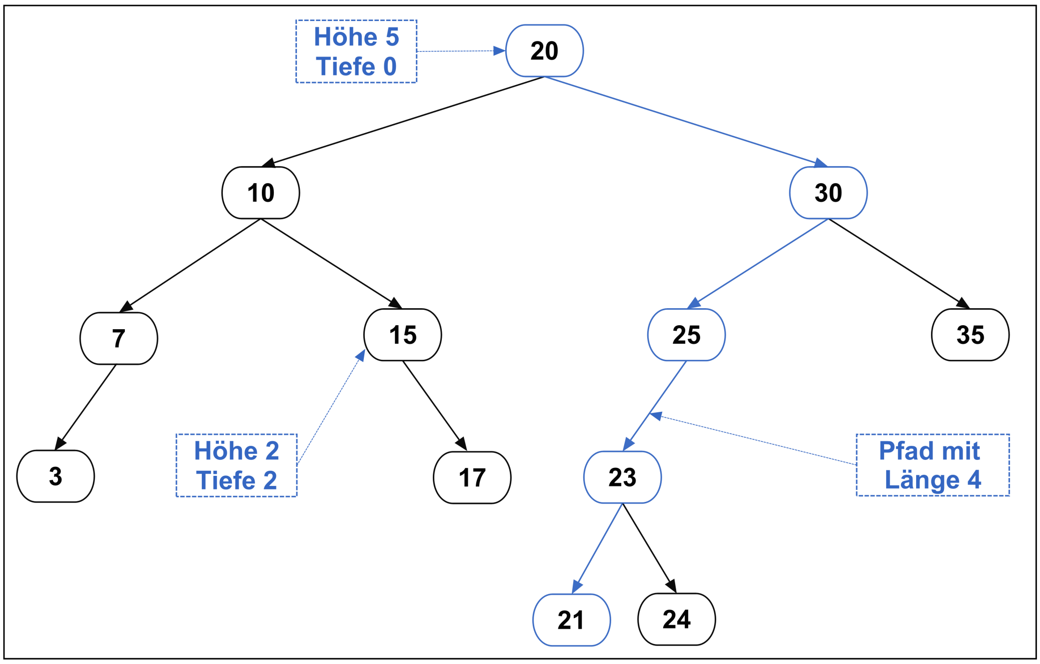

Abbildung 3: Ein weiterer binärer Suchbaum

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.

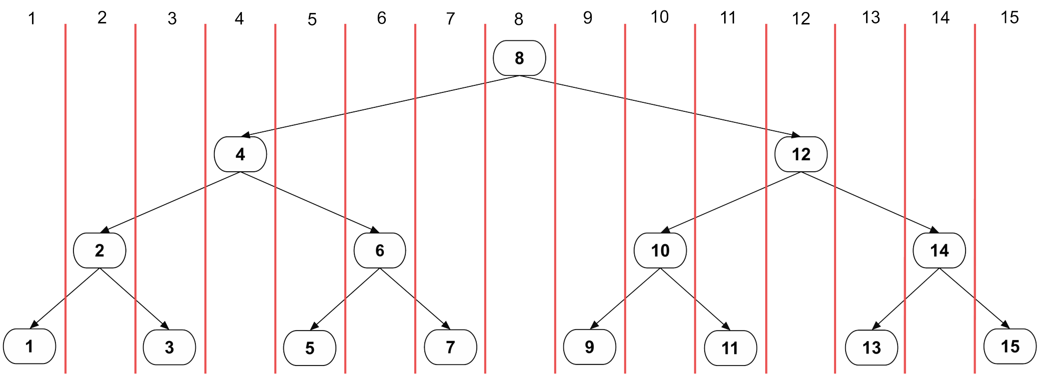

Abbildung 4: Schlüssel sind aufsteigend sortiert ablesbar.

Algorithmisch kann man sich die im BST enthaltenen Schlüssel aufsteigend sortiert durch eine **Inorder-Traversierung** ausgeben lassen. Es ist ein rekursives 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 Knotenanzahl $n + 1$ 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. Anderenfalls 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 linkem 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.

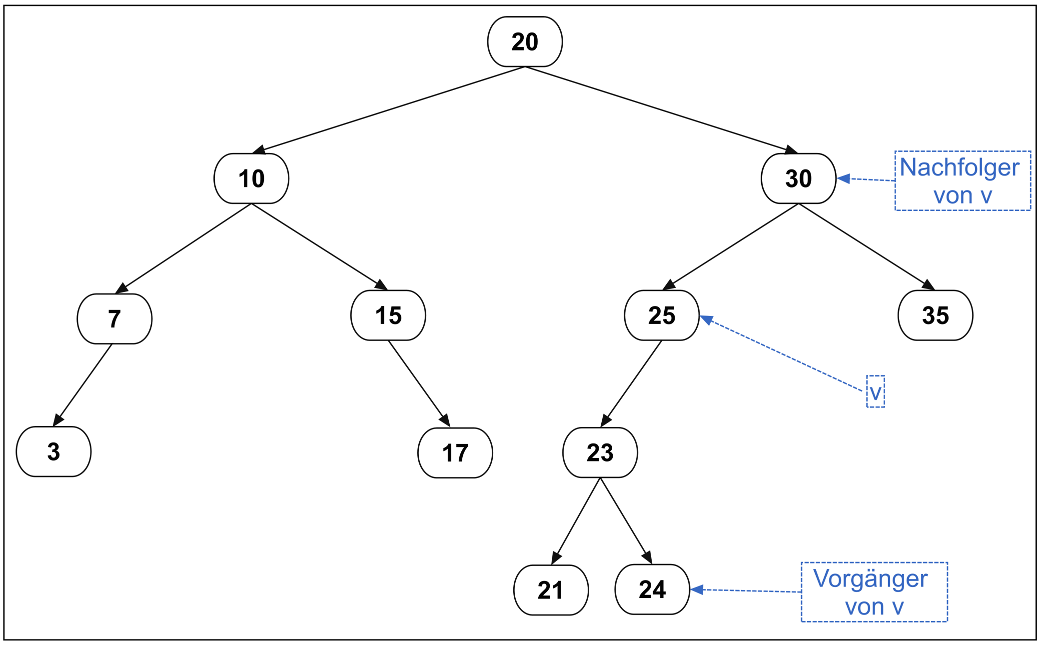

Abbildung 5: Darstellung von Vorgänger und Nachfolger.

Für alle a,b,$c \in M$ gilt:

1. $(a, a) \in R$ (reflexiv)
2. $(a, b) \in R \wedge (a, b) \in R \Rightarrow a = b$ (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-Beziehungäuf 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 Eigenschaften 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 betrachtetem 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

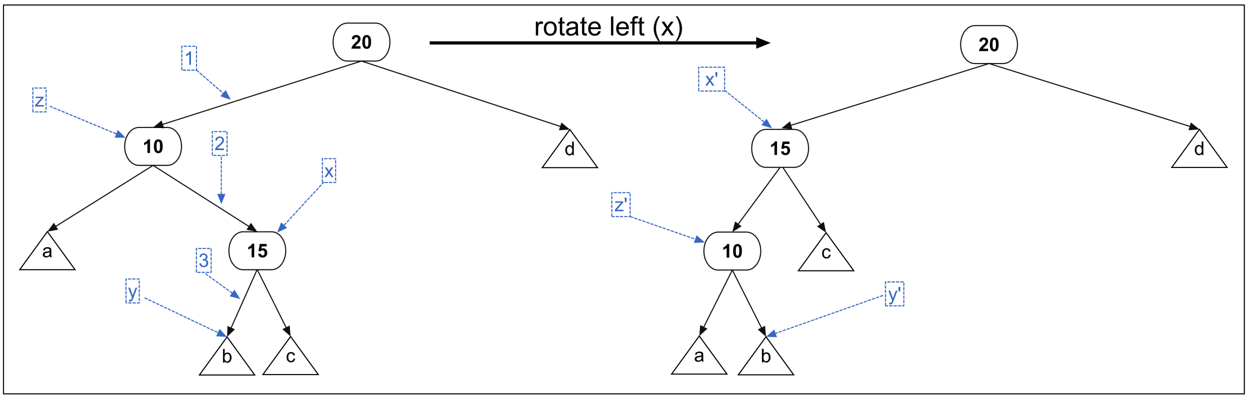

Abbildung 6: Linksrotation auf Knoten x.

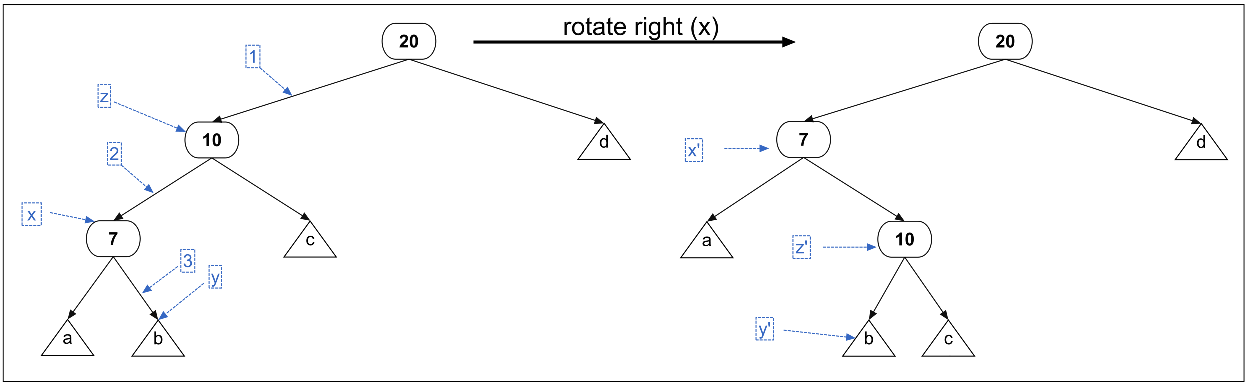

Abbildung 7: Rechtsrotation auf Knoten x.

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 eine Suche nach k ausgeführt. Findet man den Schlüssel wird das Einfügen abgebrochen und der BST bleibt unverändert. Wurde 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 zunächst eine Suche nach diesem ausgeführt. Ist k im BST nicht vorhanden wird abgebrochen und der BST bleibt unverändert. Ansonsten werden drei Fälle unterschieden.

1. v ist ein Blatt:
   v kann ohne weiteres aus dem BST entfernt werden.

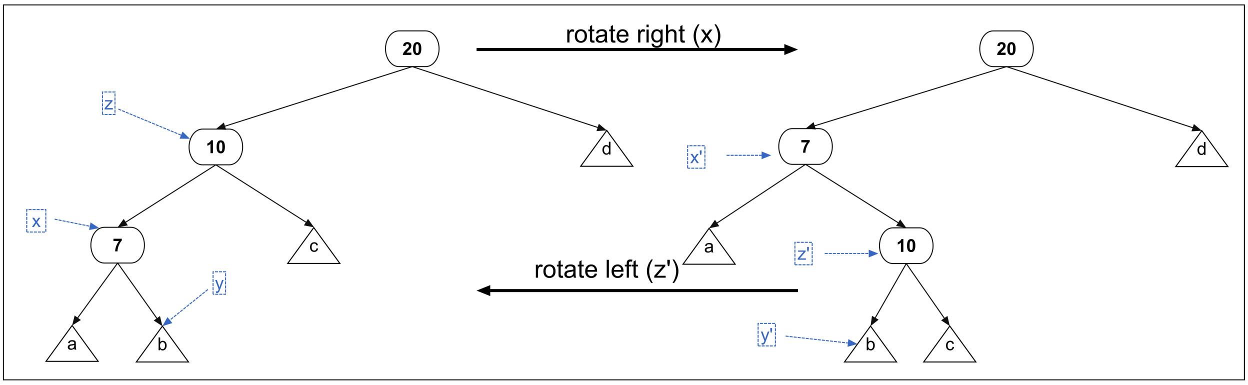

Abbildung 8: Gegenseitiges aufheben von Rotationen

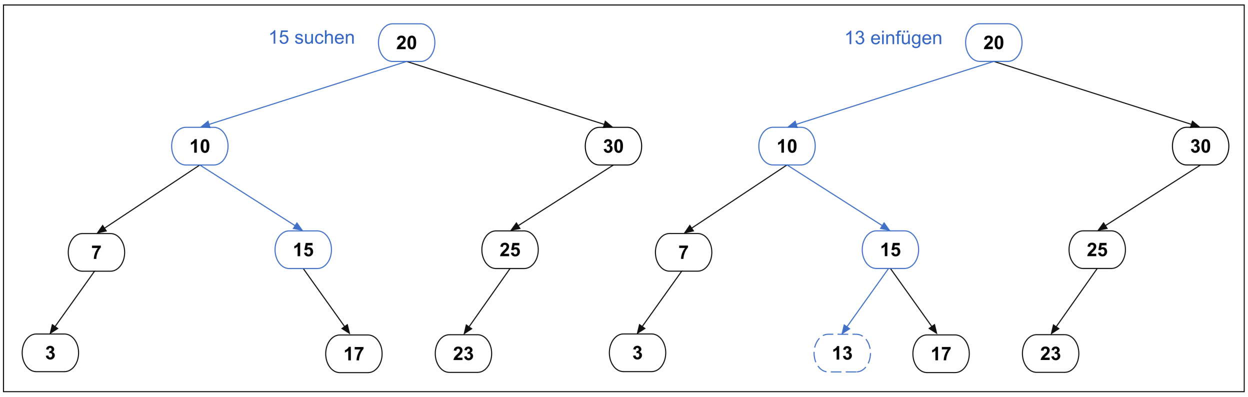

Abbildung 9: Links zeigt eine Suche nach dem Schlüssel 15. Rechts das Einfügen des Schlüssels 13

2. v hat genau ein Kind c:
   Ist v die Wurzel kann er ohne weiteres entfernt werden. 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, T_l wird links an z angefügt und T_r rechts.

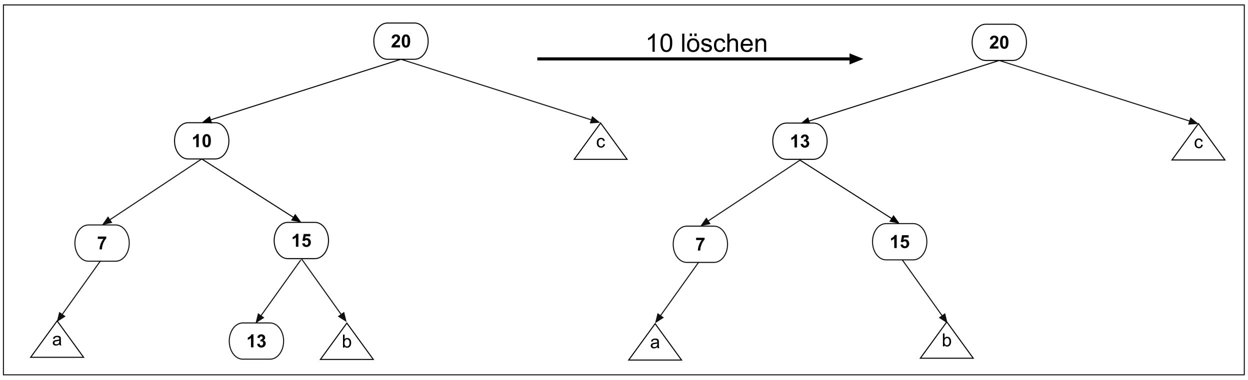

Abbildung 10: Löschen des Schlüssels 10

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:

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

$\square$

Literatur