UNIVERSITÄT BONN

Algorithmen und Programmierung

Algorithmen II

Dr. Felix Jonathan Boes
boes@cs.uni-bonn.de
Institut für Informatik

KURZE WIEDERHOLUNG

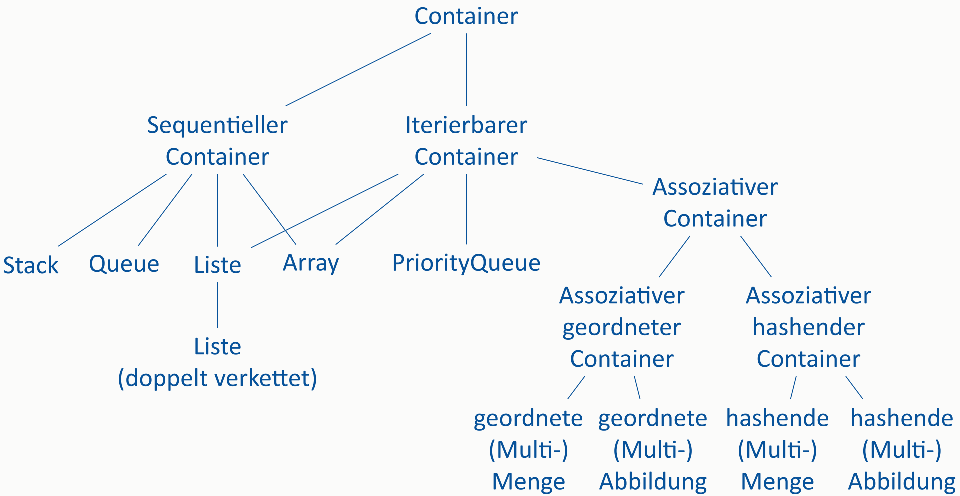
Abstrakte Datentypen und Datenstrukturen

Offene Fragen

Wie sind die wichtigsten abstrakten Datentypen definiert?

Durch welche Datenstrukturen werden sie realisiert?

Die (wichtigsten) abstrakten Datentypen im Überblick



Abstrakte Datentypen und Datenstrukturen

Assoziative Container

Offene Fragen

Was sind assoziative Container?

Durch welche Datenstrukturen werden sie realisiert?

Assoziative Container

Im Rahmen von assoziativen Containern bezeichnen wir konstante Objekte als **Keys** und nicht notwendigerweise konstante Objekte als **Values**.

Ein **assoziativer Container** ist ein iterierbarer Container zum Speichern von Keys / Key-Value-Paaren desselben Typs, der folgende Bedingungen erfüllt.

- Es ist möglich einen beliebigen Key / Key-Value-Paar einzufügen. Die durchschnittliche Laufzeit liegt in $\mathcal{O}(\log(n))$.
- Durch die Angabe eines Keys kann geprüft werden, ob dieser Key / ein zugehöriges Key-Value-Paar enthalten ist. Die durchschnittliche Laufzeit liegt in $\mathcal{O}(\log(n))$.
- Durch die Angabe eines Keys kann dieser Key / ein zugehöriges Key-Value-Paar entfernt werden (falls vorhanden). Die durchschnittliche Laufzeit liegt in $\mathcal{O}(\log(n))$.

Abstrakte Datentypen und Datenstrukturen

Assoziative Container und deren Realisierung - Geordnete Menge

Offene Fragen

Was ist eine geordnete Menge?

Durch welche Datenstruktur wird sie realisiert?

Geordnete (Multi)menge (moralisch)

Eine **geordnete Menge / geordnete Multimenge** ist ein assoziativer Container zum Speichern von **untereinander vergleichbaren** Keys. Dabei darf jeder Key höchstens einmal / mehrfach gespeichert werden.

Geordnete (Multi)menge (formaler)

Eine Menge X heißt **total geordnet** bezüglich einer binären Relation $\preceq$, falls $\preceq$ reflexiv, transitiv, antisymmetrisch und total ist.

Eine **geordnete Menge** / **geordnete Multimenge** ist ein assoziativer Container zum Speichern von total geordneten Keys. Dabei darf jeder Key höchstens einmal / mehrfach gespeichert werden.

Zur Realisierung

Geordnete Multimengen werden durch (eine Variante) von Binärbaumen realisiert. Typische Realisierungen verwenden AVL-Bäume oder Rot-Schwarz-Bäume. In diesem Modul führen wir AVL-Bäume ein.

Haben Sie Fragen?

AVL Bäume

Wir beantworten hier:

Wie wird eine geordnete Menge realisiert?

AVL Bäume

Binäre Suchbäume

Vorbereitung

Binäre Suchbäume speichern total geordnete Mengen

Binäre Suchbäume sind im Wortcase nicht effizient genug, um geordnete Mengen zu realisieren

Sie dienen als Zwischenschritt, um geordnete Mengen zu realisieren

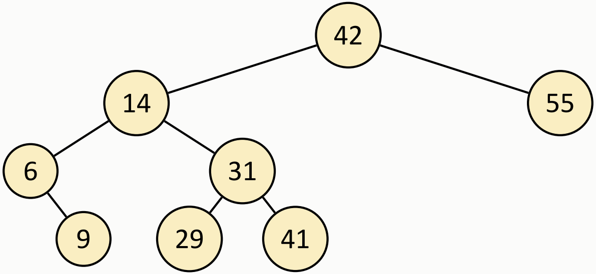
Binäre Suchbäume

(Binärer Suchbaum)

Ein (binärer) **Suchbaum** ist ein Binärbaum G, in dem für jeden Knoten k gilt:

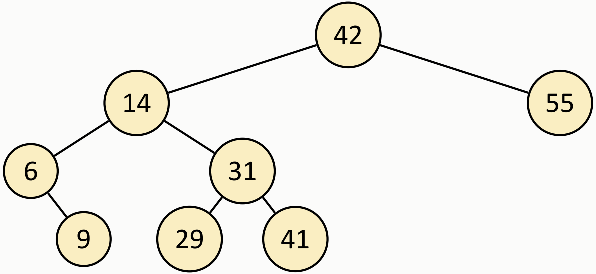
(1) der Wert von k ist größer als die Werte aller Knoten im linken Teilbaum G_l von k

(2) der Wert von k ist kleiner als die Werte aller Knoten im rechten Teilbaum G_r von k



Suchbäume und Inorderdurchlauf

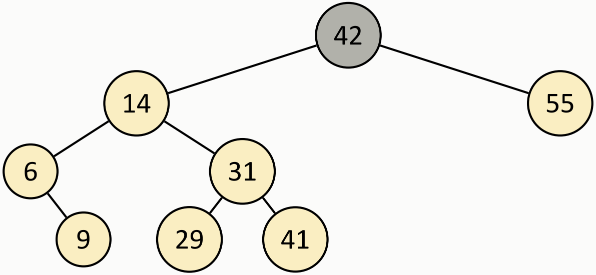
Binäre Suchbäume haben die Eigenschaft, dass die **Elemente** beim **Inorderdurchlauf aufsteigend sortiert** ausgegeben werden.



Beispielsweise liefert hier der Inorderdurchlauf: 6, 9, 14, 29, 31, 41, 42, 55.

Elemente einfügen

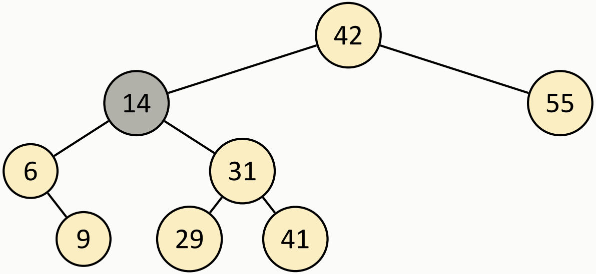
Neue Elemente werden als als Blatt hinzugefügt. Wir beginnen bei der Wurzel und entscheiden iterativ: Ist der aktuelle Knoten kleiner? Dann gehe zum linken Kind und sonst zum rechten Kind. Beispielsweise fügen wir das Element 30 hinzu.



Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet.

Elemente einfügen

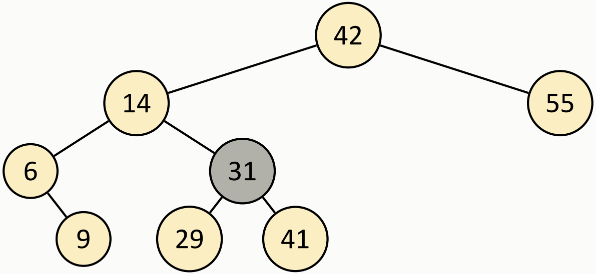
Neue Elemente werden als als Blatt hinzugefügt. Wir beginnen bei der Wurzel und entscheiden iterativ: Ist der aktuelle Knoten kleiner? Dann gehe zum linken Kind und sonst zum rechten Kind. Beispielsweise fügen wir das Element 30 hinzu.



Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet.

Elemente einfügen

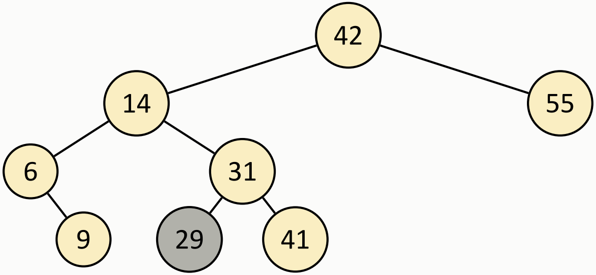
Neue Elemente werden als als Blatt hinzugefügt. Wir beginnen bei der Wurzel und entscheiden iterativ: Ist der aktuelle Knoten kleiner? Dann gehe zum linken Kind und sonst zum rechten Kind. Beispielsweise fügen wir das Element 30 hinzu.



Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet.

Elemente einfügen

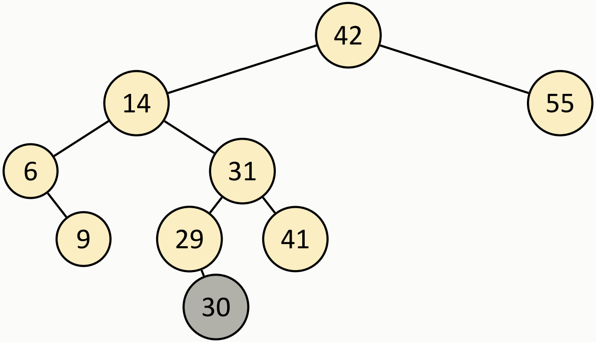
Neue Elemente werden als als Blatt hinzugefügt. Wir beginnen bei der Wurzel und entscheiden iterativ: Ist der aktuelle Knoten kleiner? Dann gehe zum linken Kind und sonst zum rechten Kind. Beispielsweise fügen wir das Element 30 hinzu.



Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet.

Elemente einfügen

Neue Elemente werden als als Blatt hinzugefügt. Wir beginnen bei der Wurzel und entscheiden iterativ: Ist der aktuelle Knoten kleiner? Dann gehe zum linken Kind und sonst zum rechten Kind. Beispielsweise fügen wir das Element 30 hinzu.



Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet.

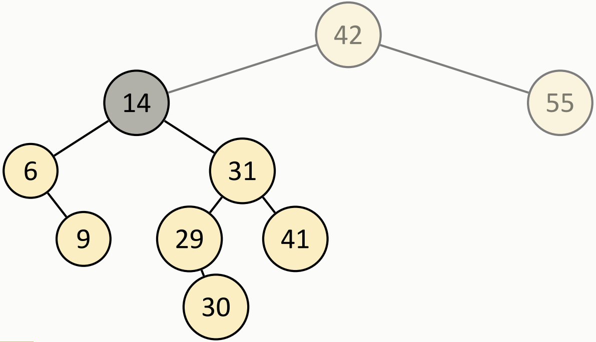
Elemente entfernen I

Ein Element wird wie folgt entfernt. Wir finden zuerst das Element und betrachten nun ausschließlich den Teilbaum mit diesem Element als Wurzel.

Falls der Teilbaum nur einen Knoten besitzt, wird dieser gelöscht und wir sind fertig.

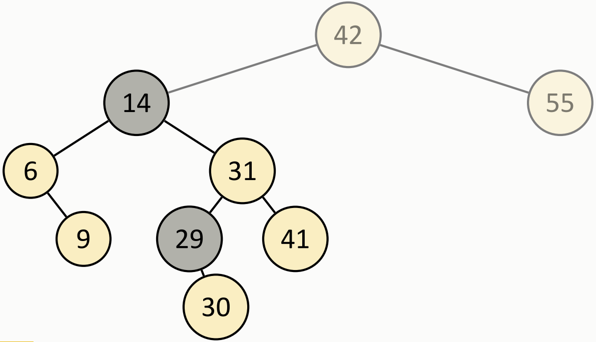
Elemente entfernen II

Andernfalls finden wir den Inorder-Nachfolger (oder Inorder-Vorgänger) der Wurzel des Teilbaums. Der Nachfolger ist das kleinste Element im rechten Teilbaum und hat deshalb selbst keinen linken Teilbaum. Die Wurzel und der Inorder-Nachfolger werden ausgetauscht und der Inorder-Nachfolger wird gelöscht. Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet. Beispielsweise löschen wir das Element 14.



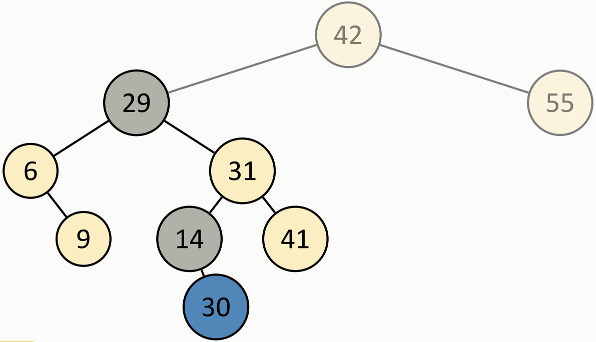
Elemente entfernen II

Andernfalls finden wir den Inorder-Nachfolger (oder Inorder-Vorgänger) der Wurzel des Teilbaums. Der Nachfolger ist das kleinste Element im rechten Teilbaum und hat deshalb selbst keinen linken Teilbaum. Die Wurzel und der Inorder-Nachfolger werden ausgetauscht und der Inorder-Nachfolger wird gelöscht. Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet. Beispielsweise löschen wir das Element 14.



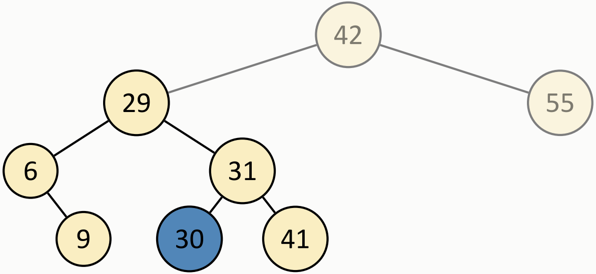
Elemente entfernen II

Andernfalls finden wir den Inorder-Nachfolger (oder Inorder-Vorgänger) der Wurzel des Teilbaums. Der Nachfolger ist das kleinste Element im rechten Teilbaum und hat deshalb selbst keinen linken Teilbaum. Die Wurzel und der Inorder-Nachfolger werden ausgetauscht und der Inorder-Nachfolger wird gelöscht. Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet. Beispielsweise löschen wir das Element 14.



Elemente entfernen II

Andernfalls finden wir den Inorder-Nachfolger (oder Inorder-Vorgänger) der Wurzel des Teilbaums. Der Nachfolger ist das kleinste Element im rechten Teilbaum und hat deshalb selbst keinen linken Teilbaum. Die Wurzel und der Inorder-Nachfolger werden ausgetauscht und der Inorder-Nachfolger wird gelöscht. Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet. Beispielsweise löschen wir das Element 14.



Zusammenfassung

Binäre Suchbäume speichern Knoten so, dass die Elemente beim Inorderdurchlauf aufsteigend sortiert sind

Beim Einfügen & Entfernen werden direkte Inordervorgägner / Inordernachfolger genutzt, um die Ordnung zu erhalten

Der Aufwand ist also $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet

Haben Sie Fragen?

AVL Bäume

Balancierte Binärbäume

Offene Frage

Wie können Suchbäume modifiziert werden, um den Aufwand von $\mathcal{O}(h)$ auf $\mathcal{O}(\log(n))$ zu verringern?

Problemstellung und Zielsetzung

Für einen Suchbaum mit n Knoten der Höhe h erfüllen alle Operationen **OP**

$$\textbf{OP} = \mathcal{O}(h)$$

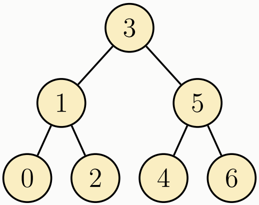
und die Höhe erfüllt

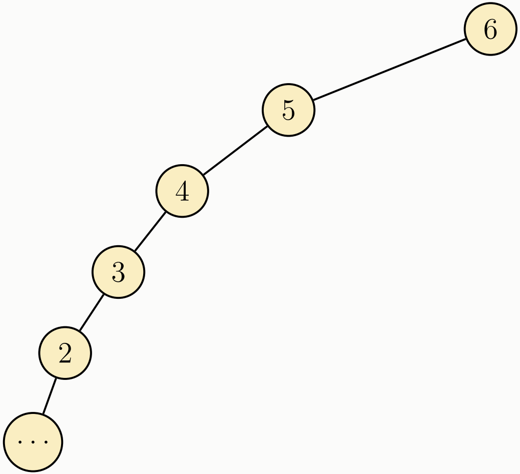
$$\log_2(n) \lesssim h \leq n.$$

Unser **Ziel** ist es, die Operationen **Insert** und **Delete** so anzupassen, dass $h = \Theta(\log(n))$ gilt. In diesem Fall sind diese „angepassten" binären Suchbäume sehr effiziente Speicherstrukturen.

Motivation I

Wir betrachten zunächst zwei Suchbäume bei denen h minimal bzw. maximal ist.





UNIVERSITÄT BONN

Motivation II

Um einem geeigneten Kriterium näher zu kommen, stellen wir uns Folgendes vor. Die Knoten sind (gleich schwere) Gewichte. An jedem Knoten ist eine Balkenwaage befestigt an welcher der linke und rechte Teilbaum aufgehangen ist.

In dieser Vorstellung hat der gesamte Graph eine geringe Höhe, wenn das linke und rechte Gesamtgewichte an jedem Knoten möglichst ausgeglichen ist.

Balancekoeffizient

(Balancekoeffizient)

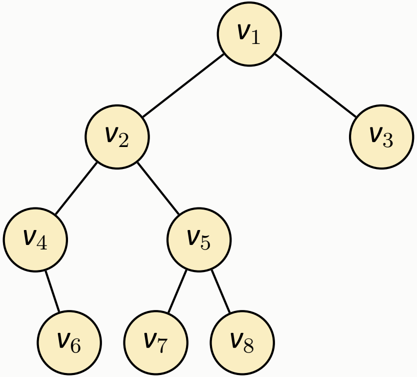
Sei G ein Binärbaum und v ein Knoten in G mit linkem Teilbaum G_l^v und rechtem Teilbaum G_r^v. Der **Balancekoeffizient** $B(v)$ ist die Höhendifferenz

$$B(v) = h(G_r^v) - h(G_l^v)\ .$$

Der Knoten v heißt **balanciert** wenn $-1 \leq B(v) \leq 1$ erfüllt ist.

Beispiel I

Wir betrachten den folgenden Binärbaum mit Wurzel v_1. Auf der rechten Seite sind die Knoten und deren Balancekoeffizient abzulesen.

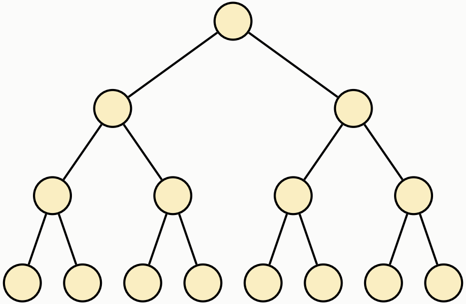


v_i	$B(v_i)$
v_1	-2
v_2	0
v_3	0
v_4	1
v_5	0
v_6	0
v_7	0
v_8	0

Beispiel II

Es gibt nur wenig Binärbäume die einen Balancekoeffizient $B(v) = 0$ für jeden Knoten v haben. Das sind genau die vollständigen Bäume. Hier gibt es pro gegebener Höhe h genau einen solchen Binärbaum.

Rechts sehen Sie einen solchen Binärbaum der Höhe $h = 3$.

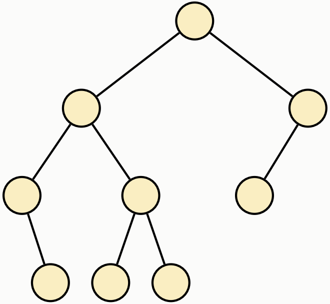


Mit diesem Beispiel sehen wir, dass die geforderte Bedingung $-1 \leq B(v) \leq 1$ nicht zu $B(v) = 0$ verschärft werden soll.

Balancierter Binärbaum

(Balancierter Binärbaum)

Ein Binärbaum ist ein **balancierter Binärbaum** wenn alle Knoten balanciert sind.



Links ist ein balancierter Binärbaum zu sehen, denn jeder Knoten ist balanciert.

Das vorangegangene **Beispiel 1** ist kein balancierter Binärbaum, da es einen Knoten gibt, der einen Balancekoeffizient von -2 hat. Dieser Knoten ist unbalanciert.

Haben Sie Fragen?

Wir zeigen:

Balancierte Binärbäume erfüllen $h = \Theta(\log(n))$

Durch Anpassen von INSERT und DELETE sind diese Operationen verträglich mit balancierten Binärbäumen

Die angepassten Operationen INSERT und DELETE sind verträglich mit Suchbäumen

AVL Bäume

Asymptotisches Wachstum der Höhe

Wie zeigen:

Bei balancierter Binärbäumen verhält sich das asymptotische Wachstum der Höhe so wie $\log(n)$

Beweisstrategie

Ein **erklärte Ziel** ist es zu zeigen, dass jeder balancierter Binärbaum der Höhe h mit n Knoten

$$\boxed{h = \Theta(\log(n))}$$

erfüllt. Wir zeigen dazu, dass für $h \geq 1$ diese zwei Ungleichungen erfüllt sind.

$$c_1 \cdot \log_2(n) \leq h \quad \text{und} \quad h \leq c_2 \cdot \log_2(n)$$

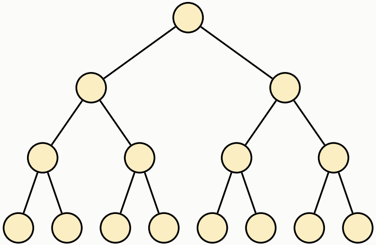
Das reicht aus, denn $\log_2$ ist ein Vielfaches von log.

Um die erste Ungleichung $c_1 \cdot \log_2(n) \leq h$ zu beweisen, fragen wir uns: Wie voll ist ein balancierter Binärbaum maximal bei gegebenem h?

Um die zweite Ungleichung $h \leq c_2 \cdot \log_2(n)$ zu beweisen, fragen wir uns: Wie voll ist ein balancierter Binärbaum minimal bei gegebenem h?

Beweis erste Ungleichung

Beobachtung: Ein Binärbaum der Höhe h ist maximal befüllt wenn er vollständig ist.



Auf jedem Level $0 \leq i \leq h$ hat der vollständige Binärbaum 2^i Knoten. Damit hat dieser Binärbaum

$$N_h = 2^0 + 2^1 + \ldots + 2^h = 2^{h+1} - 1$$

viele Knoten. Dieser Binärbaum ist auch ein balancierter Binärbaum. Also hat ein balancierter Binärbaum der Höhe h maximal $N_h = 2^{h+1} - 1$ Knoten. Es folgt

$$\log_2(n) \leq \log_2(N_h) < \log_2(2^{h+1}) = h + 1$$

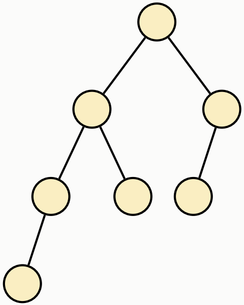
und somit

$$\boxed{\frac{1}{2} \cdot \log_2(n) \leq \frac{h+1}{2} \leq h}$$

Haben Sie Fragen?

Beweisskizze zweite Ungleichung I

Wir bezeichnen mit n_i die minimale Knotenanzahl eines balancierten Binärbaums der Höhe i. Einen zugehörigen balancierten Binärbaum nennen wir G^i. Der Balancekoeffizient der Wurzel w hängt nur von der Höhe des linken Teilbaums G^i_l und des rechten Teils G^i_r ab.



Die beiden Teilbäume sind balanciert mit Höhe $i-1$ und / oder $i-2$. Beide haben minimale Knotenanzahlen. Wir können annehmen, dass G^i_l Höhe $i-1$ sowie G^i_r Höhe $i-2$ hat.

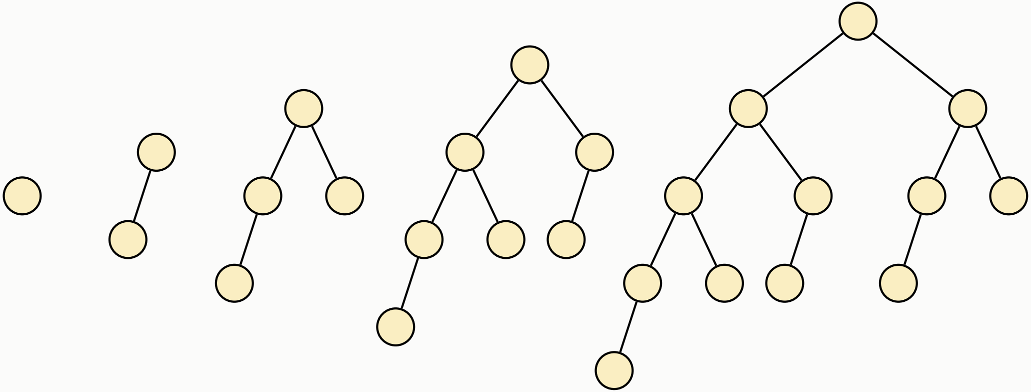
Aus dieser Beobachtung folgen zwei Aussagen.

Zum einen erhalten wir eine Konstruktionsvorschrift um balancierte Binärbäume mit minimaler Knotenzahl zu konstruieren.

Zum anderen erhalten wir eine Formel um n_i zu berechnen.

Beweisskizze zweite Ungleichung II

Wir beginnen mit einem Knoten G^0 und zwei verbundenen Knoten G^1. Wir konstruieren G_i indem wir G_{i-1} als linken Teilbaum und G_{i-2} als rechten Teilbaum der Wurzel von G_i zusammenfassen.



Beweisskizze zweite Ungleichung III

Da G_i aus G_{i-1}, G_{i-2} und einem neuen Wurzelknoten entsteht gilt die Formel:

$$n_i = 1 + n_{i-1} + n_{i-2}$$

Man sieht am vorangegangenen Beispiel

$$n_2 = 1 + 1 + 2 = 4 \geq 2 = 2^{\frac{2}{2}}$$

$$n_3 = 1 + 2 + 4 = 7 \geq 2 \cdot \sqrt{2} = 2^{\frac{3}{2}}$$

$$n_4 = 1 + 4 + 7 = 12 \geq 4 = 2^{\frac{4}{2}}$$

Induktiv folgt also aus $n_{i-1} \geq 2^{\frac{i-1}{2}}$ und $n_{i-2} \geq 2^{\frac{i-2}{2}}$:

$$n_i = 1 + n_{i-1} + n_{i-2} \geq 1 + 2^{\frac{i-1}{2}} + 2^{\frac{i-2}{2}} > 2^{\frac{i-2}{2}} + 2^{\frac{i-2}{2}} = 2^{\frac{i}{2}}$$

Deshalb gilt für einen balancierten Binärbaum der Höhe $h \geq 1$ mit n Knoten:

$$\boxed{\log_2(n) \geq \log_2(n_h) \geq \log_2(2^{\frac{h}{2}}) = \frac{h}{2}}$$

Haben Sie Fragen?

Beweisskizze

Wir haben oben gezeigt dass gilt:

$$\frac{1}{2} \cdot \log_2(n) \leq h \quad \text{und} \quad h \leq 2 \cdot \log_2(n)$$

Aus den Rechengesetzen für Exponentialfunktionen und Logarithmen folgt für den konstanten Wert $c = \log_2(e)$:

$$\log_2(n) = \log_2(e) \cdot \log(n) = c \cdot \log(n)$$

Zusammengefasst gibt es also Konstanten $c_1 = \frac{c}{2}$ und $c_2 = 2 \cdot c$ mit denen diese Ungleichungen erfüllt sind:

$$c_1 \cdot \log(n) \leq h \leq c_2 \cdot \log(n)$$

Wir haben also unser erstes Ziel erreicht:

$$\boxed{h = \Theta(\log(n))}$$

Zwischenfazit

Wir haben gezeigt, dass sich bei balancierten Binärbäumen das asymptotische Wachstum der Höhe so wie $\log(n)$ verhält

Haben Sie Fragen?

AVL Bäume

Knoten hinzufügen und entfernen

Nächstes Ziel

Wir wollen balancierte Binärbäume als Suchbäume nutzen

Um balancierten Binärbäumen Blätter hinzuzufügen und zu entfernen passen wir die bekannten Operationen INSERT und DELETE an

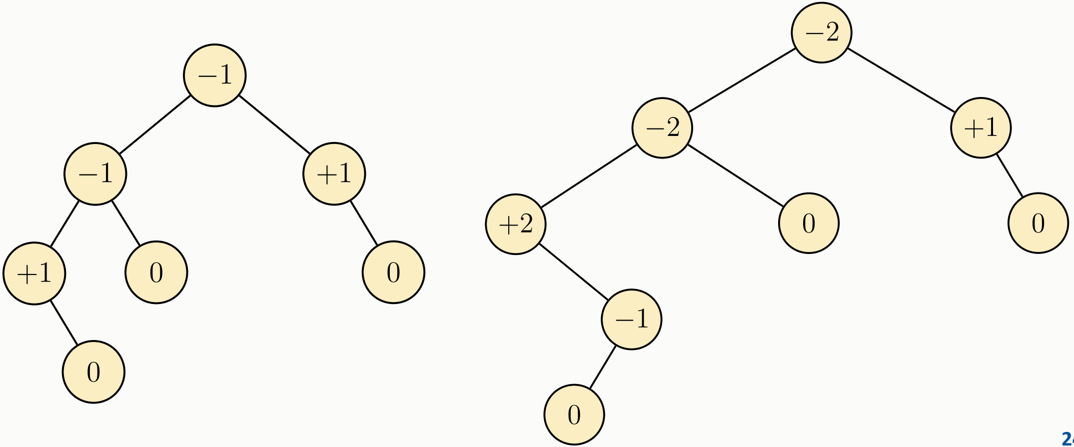
Problemstellung und Zielsetzung

Unser **Ziel** ist es balancierte Binärbäume als Suchbäume zu nutzen. Bei Suchbäumen werden schlussendlich nur Blätter hinzugefügt oder Knoten mit höchstens einem Kind entfernt. Wenn aus einem balancierten Binärbaum solche Knoten entfernt oder Blätter hinzugefügt werden, können Knoten unbalanciert werden.

Wir verstehen zuerst welche Effekte beim Einfügen und Entfernen von Blättern entstehen können. Im Anschluss passen wir das bekannte Einsetzen und Entfernen an, um Operationen zu erhalten, die balancierte Binärbäume in balancierte Binärbäume transformieren.

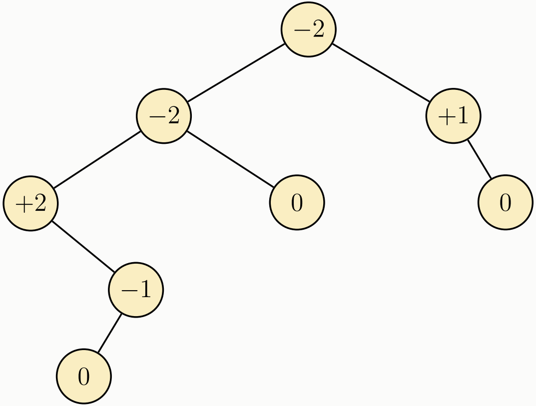
Ein erstes Beispiel

Wir betrachten zunächst einen balancierten Binärbaum, tragen die Balancekoeffizienten in den Knoten ein und fügen ein neues Blatt hinzu.



Beobachtung I

Der Balancekoeffizient eines Knoten hängt nur von der Höhe seiner beiden Teilbäume ab. Deshalb beeinflusst das Hinzufügen oder Entfernen eines Blatts nur die Vorfahren des Blatts.



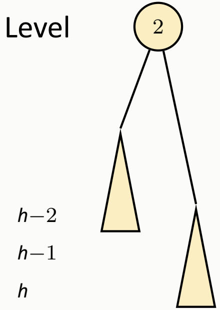
Nach dem Entfernen oder Hinzufügen eines Blatts erhalten wir dann entweder einen balancierten Binärbaum. In diesem Fall ist nichts weiteres zu tun.

Im anderen Fall gibt ein Blatt v sodass dessen Vorfahren einen Balancekoeffizient von $-2 \leq B \leq 2$ haben sowie alle anderen Knoten einen Balancekoeffizient von $-1 \leq B \leq 1$ haben.

Beobachtung II

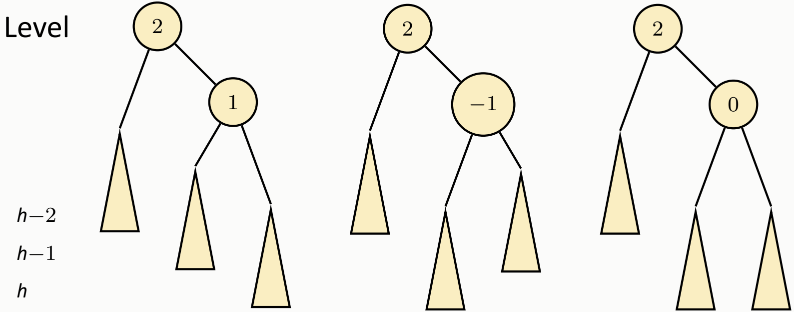
Wir konzentrieren uns auf den spannenden, zweiten Fall. Hier konzentrieren wir uns auf den höchsten Knoten der $B = \pm 2$ erfüllt. Der Einfachheit halber konzentrieren wir uns auf $B = +2$.

Der linke und rechte Teilbaum wird jeweils durch ein Dreieck angedeutet.



Beobachtung III

Hier sind drei Fälle möglich. Die zugehörigen Teilbäume werden durch Dreiecke angedeutet.

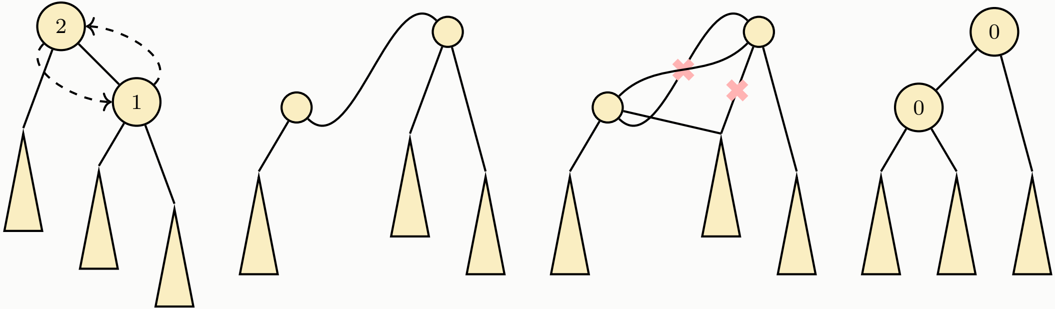


Fälle 1 und 2 treten beim Hinzufügen auf. Fälle 1, 2 und 3 treten beim Entfernen auf. Wir untersuchen zunächst Fall 1 und 2 beim Hinzufügen eines Blatts.

Haben Sie Fragen?

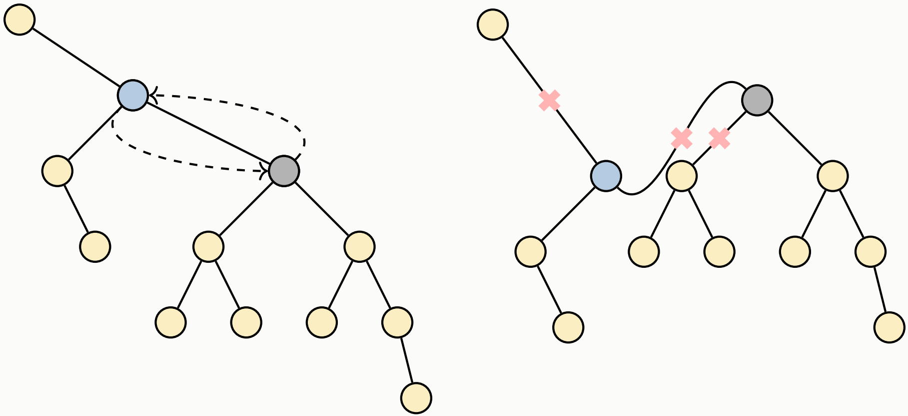
Korrektur im Fall 1

Um den Defekt im Fall 1 zu korrigieren, führen wir eine **Linksrotation** durch.

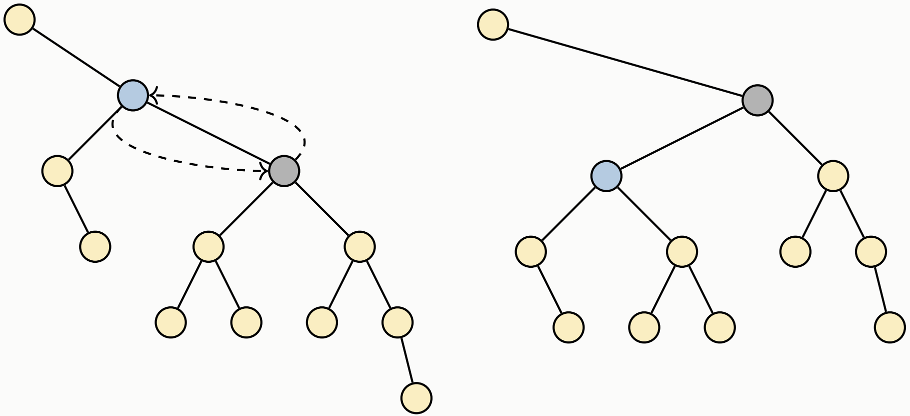


Das Hinzufügen des Blatts erhöht den Balancekoeffizient aller Vorfahren des Blatts um eins. Das Rotieren verringert den Balancekoeffizient aller Vorfahren des Knoten um den gereht wird um eins. Wir erhalten somit wieder einen balancierten Binärbaum.

Beispiel



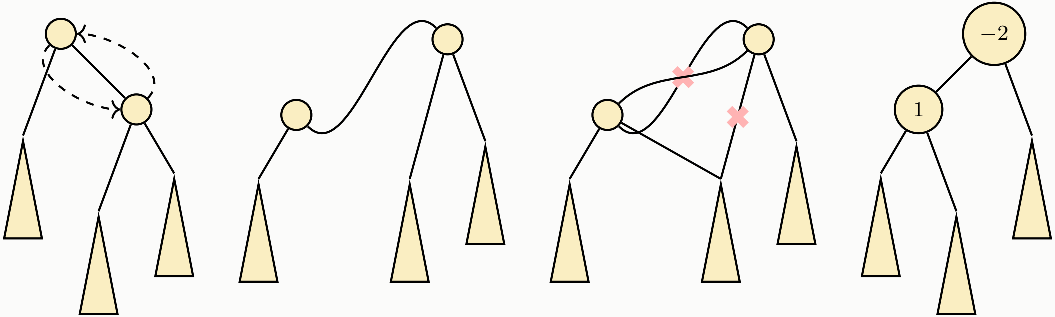
Beispiel



Haben Sie Fragen?

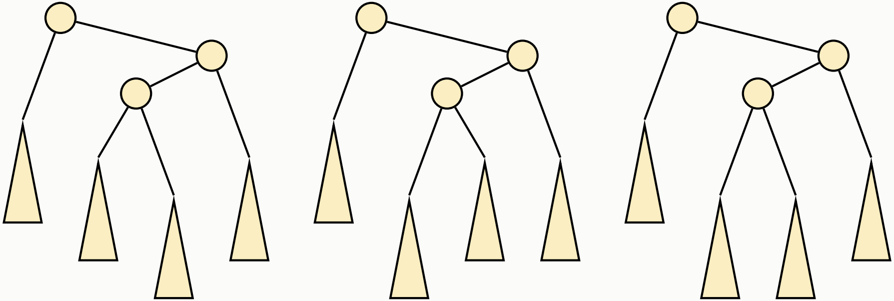
Korrektur im Fall 2

Um den Defekt im Fall 2 zu korrigieren, ist eine Rotation wie oben nicht ausreichend:



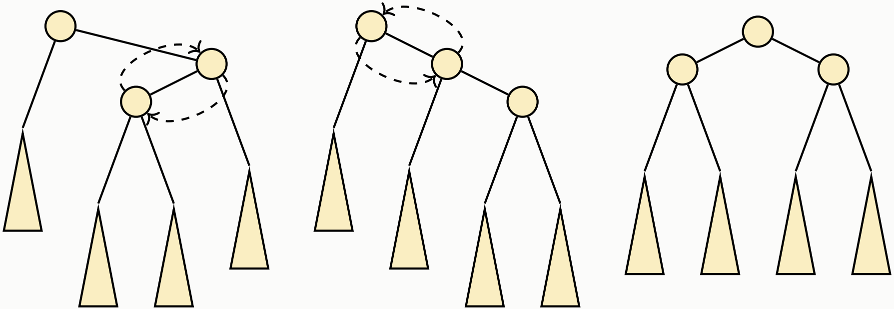
Korrektur im Fall 2

Um den Defekt im Fall 2 zu korrigieren, führen wir eine **Doppelrotationen** durch. Hier gibt es drei Unterfälle.



Korrektur im Fall 2

Wir führen eine **Doppelrotation** für den dritten Unterfall durch. Die Doppelrotation (und die Schlussfolgerung) ist für die anderen beiden Unterfälle identisch.



Das Hinzufügen des Blatts erhöht den Balancekoeffizient aller Vorfahren des Blatts um eins. Das Rotieren verringert den Balancekoeffizient aller Vorfahren des oberen Knoten um den gereht wird um eins.

Haben Sie Fragen?

Zwischenfazit

Durch das Hinzufügen eines Blatts können die Balancekoeffizienten auf dem Weg von der Wurzel zum Blatt gestört werden.

In dem Fall werden alle gestörten Balancekoeffizienten durch eine einzige (Doppel)rotation korrigiert.

Haben Sie Fragen?