Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

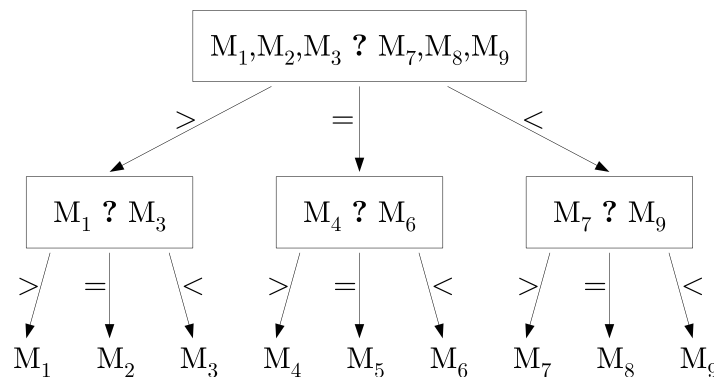
Institut für Theoretische Informatik
Peter Widmayer
Tobias Pröger
Thomas Tschager

18. März 2015

Datenstrukturen & Algorithmen — Lösungen zu Blatt 4 — FS 15

Lösung 4.1 *Untere Schranken / Algorithmenentwurf.*

a) Seien die Münzen $M_1, \ldots, M_9$ gegeben. Der folgende Ablaufbaum beschreibt eine Strategie, die in jedem Fall mit zwei Wägungen die falsche Münze ermittelt. In jedem Knoten ist vermerkt, welche Münzen in den Schalen links und rechts liegen. Nach jeder Wägung gibt es drei mögliche Ausgänge: Die Münzen links sind leichter ($<$), gleich schwer ($=$) oder schwerer ($>$) als die Münzen rechts. Nach jedem Schritt befindet sich die falsche Münze in der Menge mit höchstem Gewicht.



b) Seien die Münzen $M_1, \ldots, M_n$ gegeben. Für $n = 1$ haben wir die falsche Münze gefunden und sind fertig. Ansonsten teilen wir die Münzen in drei Gruppen $G_1 := \{M_1, \ldots, M_{n/3}\}$, $G_2 := \{M_{n/3+1}, \ldots, M_{2n/3}\}$ und $G_3 := \{M_{2n/3+1}, \ldots, M_n\}$ ein, legen G_1 in die linke Schale und G_2 in die rechte Schale. Sind die Münzen links schwerer, dann befindet sich die gesuchte Münze in G_1 und wir fahren mit dieser Menge fort. Analog wird mit G_2 fortgefahren wenn die rechte Seite schwerer ist, oder mit G_3, falls beide Seiten gleich schwer sind.

Es verbleibt zu zeigen, dass auf diese Weise wirklich nur $\log_3(n)$ Wägungen anfallen. Dazu zeigen wir mit vollständiger Induktion über $m \in \mathbb{N}$ die folgende Aussage: Für $n = 3^m$ benötigt das obige Verfahren genau m Wägungen, um die falsche Münze zu finden.

Induktionsverankerung ($m = 1$): Für $m = 1$ ist $n = 3$, und es wird genau eine Wägung benötigt um die falsche Münze zu finden.

Induktionsannahme: Sei die Aussage wahr für m, d.h. für $n = 3^m$ benötige das obige Verfahren genau m Wägungen, um die falsche Münze zu finden.

Induktionsschluss ($m \to m + 1$): Es sei $n = 3^{m+1}$. Wir teilen die Münzen in drei gleich grosse Gruppen ein und fahren mit der schwersten Menge fort. Diese enthält genau

$$n/3 = 3^{m+1}/3 = 3^m \tag{1}$$

viele Münzen, und nach Induktionsannahme genügen m Wägungen, um die falsche Münze zu finden. Damit benötigt das Verfahren insgesamt nur $m + 1$ Wägungen.

Wir beobachten nun, dass für $m = \log_3(n)$ grundsätzlich $n = 3^m$ gilt, also werden nur $\log_3(n)$ Wägungen benötigt.

c) Seien die Münzen $M_1, \ldots, M_n$ gegeben. Jeder mögliche Algorithmus kann durch einen Ablaufbaum wie auf der vorigen Seite beschrieben werden. Die maximale Tiefe T dieses Baums entspricht der maximalen Anzahl Wägungen im schlimmsten Fall. Wir müssen also zeigen, dass die Tiefe T *jedes* gültigen Ablaufbaums mindestens $\log_3(n) - 1$ beträgt.

Dazu überlegen wir zunächst, dass jede Wägung nur drei mögliche Ausgänge besitzt. Also ist jeder gültige Ablaufbaum ternär, d.h. jeder Knoten hat maximal drei Nachfolger. Wir zeigen nun zunächst mit vollständiger Induktion, dass in Tiefe k maximal 3^k Knoten existieren.

Induktionsverankerung ($k = 0$): In Tiefe 0 existiert nur ein einziger Knoten, nämlich die Wurzel des Baums.

Induktionsannahme: Wir nehmen an, die Behauptung sei wahr für k, d.h. in Tiefe k existieren maximal 3^k Knoten.

Induktionsschluss ($k \to k+1$): Betrachte die Knoten in Tiefe $k+1$. Diese können nur über eine Kante von Knoten in Tiefe k erreicht werden. Jeder Knoten besitzt maximal drei Nachfolger, und nach Induktionsannahme existieren maximal 3^k Knoten in Tiefe k. Also gibt es maximal $3 \cdot 3^k = 3^{k+1}$ Knoten in Tiefe $k + 1$.

In manchen Knoten des Ablaufbaums wird der Algorithmus ein Ergebnis ausgeben, da die vorliegenden Informationen ausreichend sind, um die falsche Münze zu identifizieren. Da jedes Ergebnis $M_1, \ldots, M_n$ möglich ist, gibt es also n mögliche Situationen, die vom Algorithmus erkannt werden müssen. Für einen korrekt arbeitenden Algorithmus muss für jede Situation mindestens ein Knoten existieren. Also muss

$$n \leq \text{Anzahl der Knoten} \leq \sum_{k=0}^{T} 3^k = \frac{3^{T+1} - 1}{2} < 3^{T+1} \tag{2}$$

gelten, und damit erhalten wir

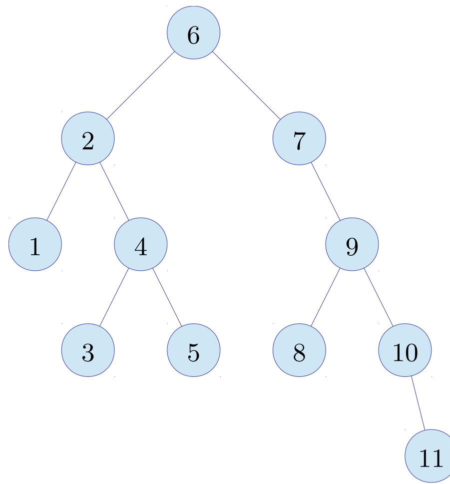
$$\log_3(n) < T + 1 \Leftrightarrow T > \log_3(n) - 1. \tag{3}$$

Die Tiefe jedes Ablaufbaums beträgt also mindestens $\log_3(n) - 1$, und damit muss jeder korrekte Algorithmus im schlimmsten Fall mindestens so viele Wägungen durchführen.

Hinweis: Man kann sich leicht überlegen, dass jedes mögliche Ergebnis des Algorithmus in einem Blatt gespeichert sein muss (und nicht in einem inneren Knoten). Weiterhin kann man leicht induktiv zeigen, dass jeder ternäre Baum der Tiefe T höchstens 3^T Blätter besitzt, folglich $n \leq 3^T \Leftrightarrow T \geq \log_3(n)$ gilt. Der beste Algorithmus muss daher sogar mindestens $\log_3(n)$ viele Wägungen ausführen, d.h., der Algorithmus aus b) ist optimal.

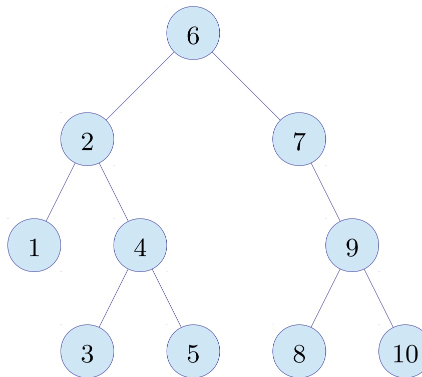
Lösung 4.2 *Natürliche Suchbäume.*

a) Es ergibt sich der folgende Baum:



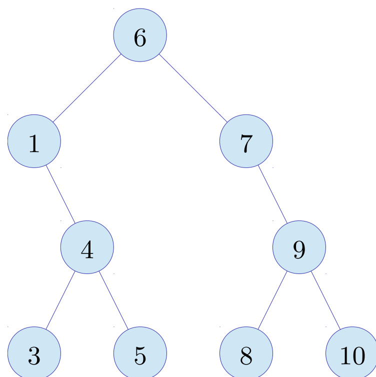
b)
- Preorder: $6, 2, 1, 4, 3, 5, 7, 9, 8, 10, 11$
- Postorder: $1, 3, 5, 4, 2, 8, 11, 10, 9, 7, 6$
- Inorder: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11$

c) Da der Schlüssel 11 in einem Blatt gespeichert ist, kann der entsprechende Knoten direkt gelöscht werden. Wir erhalten damit den folgenden Baum.

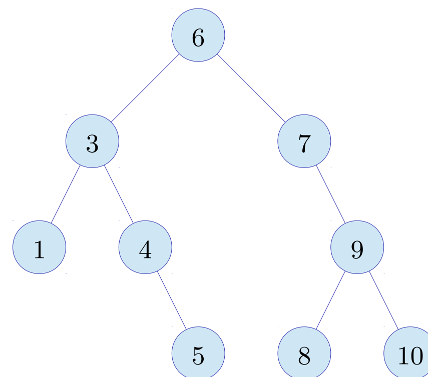


Um den Schlüssel 2 zu löschen, ersetzen wir zunächst 2 durch den Schlüssel des *symmetrischen Vorgängers* (d.h., durch den grössten Schlüssel kleiner als 2) oder durch den Schlüssel des *symmetrischen Nachfolgers* (d.h., durch den kleinsten Schlüssel grösser als 2) und löschen den entsprechenden Vorgänger- oder Nachfolgerknoten.

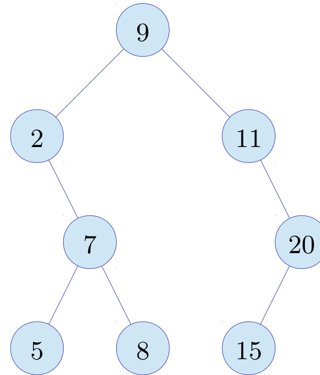
Bei Verwendung des symmetrischen Vorgängers:



Bei Verwendung des symmetrischen Nachfolgers:



d) Ein binärer Suchbaum kann aus seiner Preorder-Reihenfolge $k_1, \ldots, k_n$ rekonstruiert werden, indem die Schlüssel $k_1, \ldots, k_n$ in genau dieser Reihenfolge in einen initial leeren binären Suchbaum eingefügt werden. Damit ergibt sich der folgende Suchbaum:



Lösung 4.3 *Erweiterte Suchbäume.*

Jeder Knoten v des Baums speichert die Anzahl g_v gerader Elemente, die sich im linken Teilbaum von v befinden.

Bei der Anfrage nach der Anzahl Elemente, die kleiner als k und geradzahlig sind, suchen wir wie gewohnt nach k. Jedes Mal, wenn man während der Suche in den rechten Teilbaum eines Knotens v geht (also im Fall, in dem k grösser ist als der Schlüssel in v), inkrementieren wir einen anfangs mit Null initialisierten Zähler um g_v bzw. um $g_v + 1$, falls der Schlüssel von v gerade ist. Wird dagegen der linke Teilbaum eines Knotens besucht, dann wird der Zähler nicht verändert. Die Suche endet schliesslich in einem Knoten v, der ein Element k' speichert. Ist $k = k'$, dann war die Suche erfolgreich und wir addieren noch ein letztes Mal g_v zum Zähler. Ansonsten ist k im Baum nicht vorhanden. In diesem Fall muss der Zähler noch um 1 erhöht werden, wenn k' gerade und kleiner als k ist.

Um die Frage zu beantworten, wie viele gerade Zahlen zwischen k_1 und k_2 der Baum speichert, benutzen wir das eben beschriebene Verfahren, um die Zahlen L_1 und L_2 von geraden Elementen kleiner k_1 bzw. k_2 zu bestimmen. Subtrahiert man nun L_1 von L_2, dann erhält man die Anzahl gerader Schlüssel, die zwischen k_1 und $k_2 - 1$ liegen. Falls k_1 gerade und im Baum vorhanden ist, müssen wir von dieser Anzahl noch 1 abziehen. Ob k_1 vorhanden ist, kann direkt während der Berechnung von L_1 festgestellt werden.

Das Einfügen eines ungeraden Elements i erfolgt wie gewohnt, und wir setzen im entsprechend neu eingefügten Knoten v den Zähler g_v auf 0. Falls i gerade ist, müssen wir zusätzlich jedes Mal, wenn wir einen Knoten u treffen, der einen grösseren Schlüssel als i enthält, g_u noch um eins erhöhen (da i in den linken Teilbaum von u eingefügt wird).

Zum Löschen eines Elements i suchen wir zunächst nach i, um zu prüfen, ob i im Baum gespeichert ist. Ist dies nicht der Fall, dann sind wir fertig. Andernfalls führen wir eine erneute Suche nach i durch. Ist i gerade, dann dekrementieren wir g_u bei jedem Knoten u, wenn wir auf der Suche nach i im linken Teilbaum von u fortfahren. Am Ende der Suche wird i in einem Knoten v gefunden. Nun unterscheiden wir drei Fälle.

1. Fall: *v ist ein Blatt.* Dann kann v direkt gelöscht werden und wir sind fertig.

2. Fall: *v hat genau einen Nachfolger.* Auch hier kann v ohne Weiteres gelöscht und durch seinen entsprechenden Nachfolger ersetzt werden.

3. Fall: *v hat zwei Nachfolger.* Dann enthält der linke Teilbaum von v den symmetrischen Vorgänger j von i. Wir ersetzen den in v gespeicherten Schlüssel durch j. Danach wird der Knoten w, der j enthält, wie gewohnt gelöscht. Ist j gerade, dann enthält der linke Teilbaum von v einen geraden Schlüssel weniger, also muss in dem Fall g_v noch dekrementiert werden.

Alle drei neuen Operationen haben einen konstanten Aufwand pro Knoten und daher keinen negativen Einfluss auf die Laufzeit der drei Operationen Einfügen, Löschen und Suchen. Die Laufzeit ist also weiterhin $\mathcal{O}(h)$, wobei h die Höhe des Baums bezeichnet.