Algorithmen und Datenstrukturen

SoSe 2008 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Algorithmen und Datenstrukturen

Gesamtübersicht

- Organisatorisches / Einführung
- Grundlagen: RAM, $\mathcal{O}$-Notation, Rekursion, Datenstrukturen
- Sortieren
- Wörterbücher und Mengen
- Graphen und Graphalgorithmen

Geordnete Mengen als Wörterbücher

Wir betrachten (wie bisher) Mengen M, allerdings mit einer totalen Ordnungsrelation $\leq$ auf dieser Menge. versehen.
Beispiel: Betrachte $M = \mathbb{Z}$ mit der üblichen Ordnung.

Die Elemente von M können wir uns (zunächst) in einem Feld $A[1..|M|]$ abgespeichert denken, und zwar *ordnungserhaltend*, d.h.: $i < j$ gdw. $A[i] < A[j]$.
(Wir können die Forderung auch noch abschwächen, falls wir zulassen wollen, dass zwei Elemente des Feldes gleich sein dürfen. Versuchen Sie, für diesen Fall “ordnungserhaltend” zu definieren.)

Wörterbücher: Mengen mit eingeschränkten Operationen

Nachschlagen (Lookup), also die Elementanfrage
Einfügen (Insert)
Löschen (Delete) eines einzelnen Elementes (musste gemäß bisheriger Spezifikation durch andere Mengenoperationen nachgebildet werden (wie genau?))
Initialisieren (init/clear)
Wir wissen bereits: Das Suchen in geordneten Feldern geht schnell (logarithmischer Zeitaufwand VL6).
Allerdings ist das Einfügen teuer (Linearzeit), da im Schnitt die Hälfte der Feldelemente verschoben werden muss.
Für nicht (oder nur sehr selten) zu verändernde geordnete Mengen ist die Darstellung durch ordnungserhaltende Felder aber nicht schlecht.

Geordnete Wörterbücher: Rekursion versus Induktion

Algorithm 1 Allgemeine Suche

Input(s): a sorted array $A : \mathbb{Z}[1..n]$, an element x
Output(s): if found: position i such that $x = A[i]$; NO otherwise

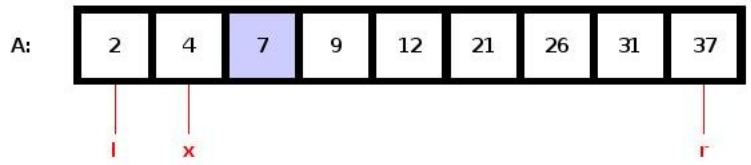
$\alpha \leftarrow 1$; $\omega \leftarrow n$;
$i \leftarrow$ some number in $[\alpha..\omega]$
while $(x \neq A[i]) \wedge (\omega > \alpha)$ **do**
  if $x < A[i]$ **then**
    $\omega \leftarrow i - 1$
  else
    $\alpha \leftarrow i + 1$
  $i \leftarrow$ some number in $[\alpha..\omega]$
end while
return i (if $A[i] = x$); NO otherwise

Algorithm 2 Allgemeine Suche rekursiv SR

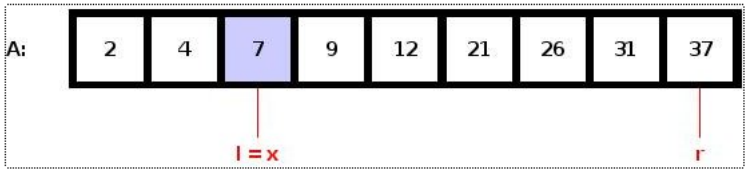
Input(s): a sorted array $A : \mathbb{Z}[1..n]$, an element x
Output(s): if found: position i such that $x = A[i]$; NO otherwise

$i \leftarrow$ some number in $[1..n]$
if $(x < A[i]) \wedge (i > 1)$ **then**
  return rescale SR$(A[1..i-1], x)$
else if $(x > A[i]) \wedge (i < n)$ **then**
  return rescale SR$(A[i+1..n], x)$
else
  return i (if $A[i] = x$); NO otherwise

Interpolationssuche am Beispiel: Wir suchen $y = 7$.



$x = 1 + \frac{7-2}{37-2} * (9-1) \approx 2,15 \rightarrow 2$ ist neuer *Teilungspunkt*



Satz: Sind die Zahlen im Feld $A[1], \ldots, A[n]$ gleichverteilt unabhängig aus dem Intervall (l, r) gezogen, so beträgt die Zugriffszeit für Interpolationssuche im mittleren Fall $\mathcal{O}(\log \log n)$.

Exkurs: Wahrscheinlichkeitsrechnung

Annahme: $x_1, \ldots, x_n$ werden unabhängig gemäß Gleichverteilung auf Intervall (x_0, x_{n+1}) gezogen.

$x \in (x_0, x_{n+1})$ ist der Vergleichswert.

$\rightsquigarrow p = \frac{x - x_0}{x_{n+1} - x_0}$ Wahrsch., dass bel. $x_j < x$ gilt.

$\rightsquigarrow$ Wahrsch., dass genau k der n Schlüssel kleiner als x sind, ist gleich

$$\binom{n}{k} \cdot p^k (1-p)^{n-k}$$

Dies ist die sog. *Binomialverteilung*, siehe VL 17 / 18 von DSL.

Betrachte ZV X, die die # der x_j angibt, die kleiner als x sind.

Aus DSL wissen wir: $\mu = pn$ ist der Erwartungswert $E[X]$.

Für die Varianz gilt: $E[(X - E[X])^2] = \sigma^2 = p(1-p)n$.

Tschebyscheffsche Ungleichung $\rightsquigarrow P[|X - \mu| \geq t] \leq \frac{\sigma^2}{t^2}$

Exkurs: Das Basler Problem

Beim Basler Problem handelt es sich um die Frage nach der Summe der reziproken Quadratzahlen:

$$\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots = 1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \cdots$$

1644 fragte sich der Italiener Pietro Mengoli, ob diese Summe konvergiere, und wenn ja, gegen welchen Wert. Er kam zu keiner Antwort. Die Basler Bernoulli-Dynastie bemühte sich auch vergeblich, 1735 fand Euler (damals in Basel) die Lösung und veröffentlichte sie in seinem Werk "De Summis Serierum Reciprocarum".

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

Die Kreiszahl π taucht auf, da die Formel, gemäß Euler, mit der Quadratur des Kreises zusammenhängt.

Die Riemannsche ζ-Funktion birgt noch viele Geheimnisse.

Schnellere Suchzugriffe in Binär(such)bäumen

Bereits erwähnt: Interpolationssuche mit mittlerer Zugriffszeit $\mathcal{O}(\log\log n)$.
Analyse ist sehr aufwändig (auch nicht in Lehrtexten enthalten).
Alternativ hier: *quadratische Binärsuche*
Grundidee (jeweils): Beschleunige Hinabsteigen im Suchbaum.
Konzeptuell kann man sich die (seltsam anmutende) Laufzeit erklären durch Binärsuche auf Zugriffspfad(en).
Kann man aber rasch einen Knoten ausfindig machen, der auf halben Wege auf dem richtigen Suchpfad liegt ?
Beobachte: Es gibt $2^{\frac{1}{2}\log_2(n)} = \sqrt{n}$ viele Knoten "auf halbem Wege".
Diese sind in der Darstellung durch ein geordnetes Feld $\sqrt{n}$ Positionen voneinander entfernt.
Zu zeigen: Lineare Suche im Abstand $\sqrt{n}$ hat im Mittel konstanten Zeitaufwand.
In $\mathcal{O}(1)$ kann man die Mitte des Suchpfades bestimmen. $\rightsquigarrow$ Aufwand $\mathcal{O}(\log\log(n))$.

Algorithm 3 Quadratische binäre Suche QBS

Input(s): a sorted array $A : \mathbb{Z}[1..n]$, an element x
Output(s): if found: position i such that $x = A[i]$; NO otherwise

$j \leftarrow 0$;
$i \leftarrow 1 + \left\lfloor \frac{x - x_0}{x_{n+1} - x_0} \cdot n \right\rfloor$
while $(i + j\sqrt{n} \leq n) \wedge (x > A[\lfloor i + j\sqrt{n} \rfloor])$ **do**
  $j++$;
end while
if $j > 0$ **then**
  return rescaled QBS$(A[\lfloor i + (j-1)\sqrt{n} + 1 \rfloor .. \lfloor i + j\sqrt{n} \rfloor], x)$
{Downwards search (from i on) analogously}
if $j > 0$ **then**
  return rescaled QBS$(A[\lfloor i - (j-1)\sqrt{n} + 1 \rfloor .. \lfloor i - j\sqrt{n} \rfloor], x)$
else
  return i (if $A[i] = x$); NO otherwise

Erinnerung: Rekursive Variante der allgemeinen Suche in geordneten Mengen ähnlich (bis auf kompliziertere Bestimmung vom Teilungspunkt i).

C: Mittlere # Vergleiche, um Teilfeld der Größe $\sqrt{n}$ zu ermitteln

p_j: Wahrsch., dass j oder mehr Vergleiche benötigt werden.
Trick aus VL11 $\rightsquigarrow C = \sum_{j\geq 1} j(p_j - p_{j+1}) = \sum_{j\geq 1} p_j$.
Aufgabe: Bestimmung der p_j. Klar: $p_1 = p_2 = 1$ (mind. zwei Vergleiche sind nötig)
$\rho(x)$: *Rang* von x in A, d.h., $|\{x_\ell \mid x_\ell < x\}|$. $\rho(x)$ ist ZV.
Werden $j > 3$ Vergleiche benötigt, gilt: $|\rho(x) - i| \geq (j-2)\sqrt{n}$.
$\rightsquigarrow p_j \leq P[|\rho(x) - i| \geq (j-2)\sqrt{n}]$.

Die mittlere Anzahl der Schlüssel, die kleiner als x sind, ist der mittlere Rang von x, ist gleich pn gemäß obiger Überlegungen zur Binomialverteilung (in denen ρ noch X genannt wurde).

Aus der Tschebyscheffschen Ungleichung folgt:

$$p_j \leq \frac{p(1-p)n}{((j-2)\sqrt{n})^2} = \frac{p(1-p)}{(j-2)^2} \leq \frac{1}{4(j-2)^2}$$

Die letzte Ungleichung folgt wegen $p(1-p) \leq 0,25$, falls $0 \leq p \leq 1$.
(Weitere Eigenschaften der *Entropiefunktion* siehe Vorl. zu Informationstheorie)
Basler Problem liefert: $C \leq 2 + \sum_{j \geq 3} \frac{1}{4(j-2)^2} = 2 + \frac{\pi^2}{24} \approx 2,4$

Ist $\bar{T}(n)$ die mittlere Anzahl der Vergleiche, die bei der Suche nach x in einem Feld mit n zufälligen Schlüsseln (gezogen aus (x_0, x_{n+1})), so gilt:
$\bar{T}(n) \leq C + \bar{T}(\sqrt{n})$, denn die Teilfelder der Größe $\sqrt{n}$ sind wiederum zufällig.
Mit $\bar{T}(1) \leq 1$ und $\bar{T}(2) \leq 2$ folgt leicht per Induktion:
$\bar{T}(n) \leq 2 + C \log\log n$ für $n \geq 2$.

Satz: Die mittleren Such-Kosten von QBS sind $\mathcal{O}(\log\log n)$.

Hinweis: Im schlechtesten Fall sind die Such-Kosten bei QBS $\mathcal{O}(\sqrt{n})$. Mit weiteren Tricks lässt sich aber das Verhalten im schlimmsten Fall auf $\mathcal{O}(\log n)$ verbessern.

Halten wir fest...

Binäre Suchbäume sind im Mittel gar nicht so schlecht.
Das Suchen in ihnne kann (im Mittel) nochmals beschleunigt werden mit Hilfe von Interpolationsideen.

> Ist es möglich, auch im schlechtesten Fall alle Baum-Operationen höchstens in logarithmischer Zeit durchzuführen?

Das würde Zweifel beseitigen, es könnte auch mal langsam gehen.

AVL-Bäume Adelson-Velskij/Landis 1962

Ein *AVL-Baum* ist ein 1-ausgeglichener Suchbaum, d.h., es handelt sich um einen binären Suchbaum, in dem sich für jeden inneren Knoten die Höhen der beiden anhängenden Teilbäume höchstens um Eins unterscheiden.

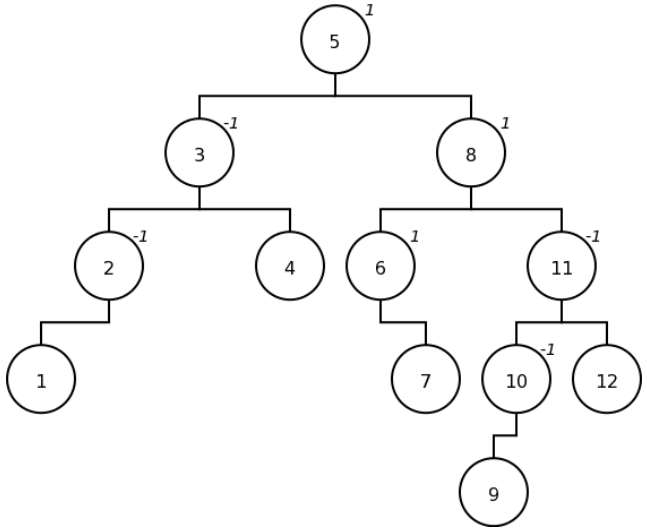
Das Suchen nach einem Element funktioniert weiter wie bei “normalen” Suchbäumen (die Suchbaumeigenschaft ist ja erfüllt).

Beim Einfügen und Löschen muss evtl. durch nachträgliches *Rebalancieren* die 1-Ausgeglichenheit wiederhergestellt werden.

Das wollen wir zunächst beobachten in: `http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm`.

AVL-Bäume: Balance

Jedem Knoten können wir als *Balancewert* die Differenz der Höhen des rechten und linken darunterhängenden Teilbaumes zuordnen:



AVL-Bäume: Einfügen

Wie üblich wird zunächst nach der Einfügestelle im Suchbaum gefahndet.
Der auf diese Weise beschriebene Suchpfad wird nach dem Einfügen rückwärts durchlaufen, um nach Knoten zu schauen, die aus der Balance geraten sind.
Wird solch ein Knoten x mit einem Balancewert von 2 oder (-2) gefunden, so muss rebalanciert werden.
Dies geschieht durch Umordnen des an x hängenden Teilbaums, was aber allenfalls die Balancewerte der Enkelknoten beeinflusst, sodass kein erneutes Absteigen nötig ist.
Dadurch kann sich die Höhe des an x hängenden Teilbaumes nicht verringern.
Deshalb müssen wir beim Einfügen nicht weiter bis zur Wurzel hochsteigen, um möglicherweise weitere Rebalancierungen durchzuführen.
O.E. diskutieren wir ein x mit Balancewert 2 auf den nächsten Folien.

AVL-Bäume: Einfügen

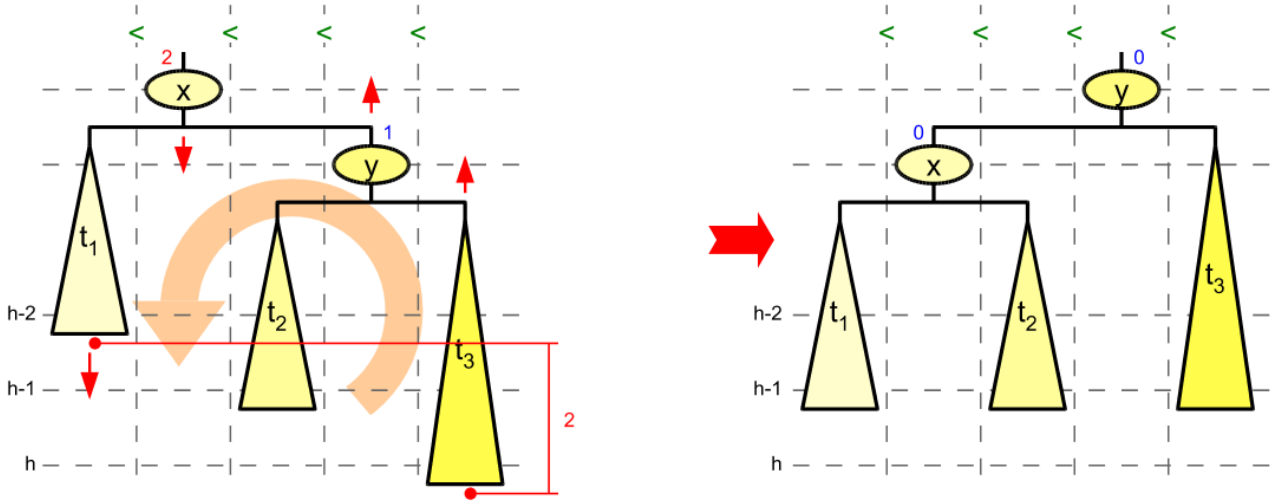
O.E. diskutieren wir ein x mit Balancewert 2. Sei y das rechte Kind von x.

1. Fall: Der rechte Teilbaum von y ist zu groß (dort Einfügestelle).

$\rightsquigarrow$ Der linke Teilbaum von y ist kleiner als der rechte, da sonst schon vor dem Einfügen Imbalance bei x bestanden hätte.

Lösen durch einfache *Rotation*.

Beachte: Suchbaumeigenschaft bleibt erhalten (Inorder-Durchlauf zum Prüfen).



AVL-Bäume: Einfügen

O.E. diskutieren wir ein x mit Balancewert 2. Sei z (!) das rechte Kind von x.

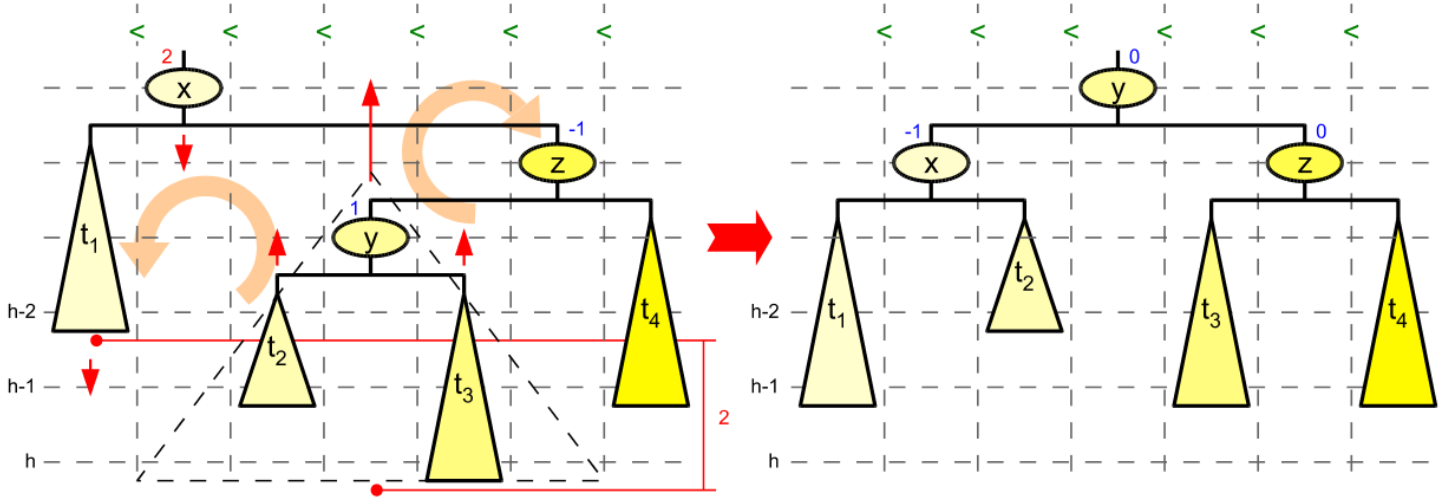
2. Fall: Der linke Teilbaum von z ist zu groß (dort Einfügestelle).

Eine einfache Rotation hülfe hier nichts; stattdessen werden weiter das linke Kind y von z (mit seinen Teilbäumen) mit einbezogen in die *Doppelrotation*.

2a Fall: Der rechte Teilbaum von y ist zu groß (dort Einfügestelle).

$\rightsquigarrow$ Der linke Teilbaum von y ist kleiner als der rechte (s.o.)

2b Fall: Der linke Teilbaum von y ist zu groß geht genauso (!)

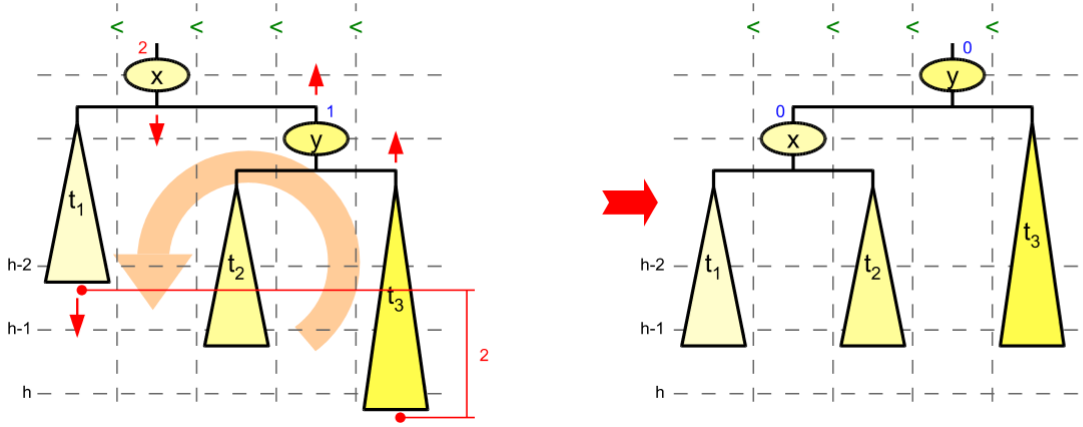


AVL-Bäume: Löschen

Wie üblich wird zunächst nach dem zu löschenden Element im Suchbaum gefahndet.
Der auf diese Weise beschriebene Suchpfad wird nach dem Einfügen rückwärts durchlaufen, um nach Knoten zu schauen, die aus der Balance geraten sind.
Betrachten wir o.E. einen Knoten x, in dessen linken Teilbaum wir einen Knoten löschen, was zu einer Nicht-1-Ausgeglichenheit führt.
Sei y das rechte Kind von x.
Ist der linke Teilbaum von y nicht höher als der rechte, so hilft einfache Rotation.
Ist der linke Teilbaum von y höher als der rechte, führt dies zu einer Doppelrotation, wobei wiederum die Enkel von x zu diskutieren sind.
(Mehr an der Tafel oder im Buch von Güting, S. 125)
Das Studieren von `http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm` sei zum weiteren Verständnis empfohlen.

AVL-Bäume: Löschen mit einfacher Rotation

Warum muss man den Suchpfad evtl. ganz zurückverfolgen ?



Offenbar entstand das Bild durch Löschen im linken Teilbaum.

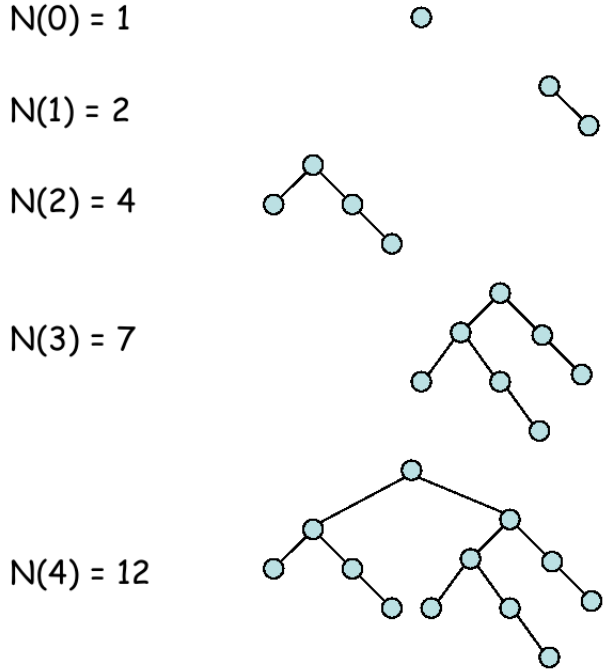
Außer die im Bild gezeigte gibt es jetzt noch die Situation, dass t_2 und t_3 gleich hoch sind. Dann ist nach der Rotation der abgebildete Teilbaum gleich hoch wie vor dem Löschen, sodass nicht weiter oben im Baum nach Balancewerten von ± 2 gesucht werden muss.

In der abgebildeten Lage jedoch verringert sich die Höhe gegenüber der vor dem Löschen, was zu neuen Balancewerten ± 2 weiter oben im Baum führen kann.

Entsprechend kann man die Doppelrotation diskutieren.

AVL-Bäume: Was bringt es?

Offenbar benötigt jede Operation $\mathcal{O}(h)$ Aufwand, wobei h die Höhe des AVL-Baumes ist.
Wie ist denn die maximale Höhe eines AVL-Baumes?
Dazu betrachten wir umgekehrt $N(h)$, die minimale Anzahl von Knoten in einem AVL-Baum der Höhe h.
Die Rekursionsgleichung legt einen Zusammenhang mit den Fibonaccizahlen nahe.



Fibonacci-Bäume sind ein gebräuchlicher Name für Bäume T_h mit $N(h)$ Knoten und Höhe h, die rekursiv aus einer Wurzel sowie links einem T_{h-1} und rechts einem T_{h-2} aufgebaut sind.
Wie soeben überlegt, sind Fibonacci-Bäume extremale AVL-Bäume.
Erinnerung: *Fibonacci-Zahlen*: $f_n = f_{n-1} + f_{n-2}$; Sonderfälle: $f(0) = 0$ und $f(1) = 1$, siehe VL14 DSL.
Bekannt: $f_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$, wobei $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6181\ldots$ die *Zahl des Goldenen Schnitts* ist. Ein weiterer Zusammenhang: $\lim_{n\to\infty} \frac{f_n}{f_{n-1}} = \phi$.
Lemma: $N(h) = f_{h+3} - 1$
Beweis: Induktion; im IS: $N(k+1) = 1 + N(k) + N(k-1) = 1 + f_{k+3} - 1 + f_{k-4} - 1 = f_{k-2} - 1$.
Hinweis: f_h ist gerade die Blattanzahl von T_h.

Fibonacci-Bäume und AVL-Bäume

Also: Ein AVL-Baum der Höhe h hat mindestens $f_{h+3} - 1$ Knoten.

$\rightsquigarrow$ Ein AVL-Baum mit n Knoten hat höchstens die Höhe h, bestimmt durch: $N(h) \leq n \leq N(h+1)$.

$\rightsquigarrow \frac{1}{\sqrt{5}}\phi^{h+3} - 2 \leq \frac{1}{\sqrt{5}}\left(\phi^{h+3} - (1-\phi)^{h+3}\right) \leq n + 1$

Logarithmieren und Umsortieren liefert: $h \leq \log_\phi(n) + c$ für eine Konstante c, d.h., $h \leq 1.4405 \log_2(n) + c$.

$\rightsquigarrow$ Lemma: Ein AVL-Baum mit n Knoten hat logarithmische Höhe; genauer ist er höchstens um 44% höher als irgendein Binärbaum mit n Knoten.

$\rightsquigarrow$ Satz: Alle (wichtigen) Operationen lassen sich auf AVL-Bäumen in Laufzeit $\mathcal{O}(\log(n))$ realisieren.

Höhenbalancierte Bäume

Wir haben mit AVL-Bäumen ein Beispiel für sogenannte *höhenbalancierte Bäume* kennengelernt.
Dies ist nicht das einzige Balance-Kriterium, wie wir in der nächsten VL sehen werden.
Es gibt viele weitere höhenbalancierte Baum-Modelle, z.B. Rot-Schwarz-Bäume.
Diese werden (gemeinsam mit AVL) in der folgenden Semesterarbeit (an der Uni Leipzig) vorgestellt: `http://leechuck.de/ginfin/baum1/index.html`
(auf den Seiten findet sich übrigens auch JAVA-Quellcode hierzu sowie ein nettes JAVA-Applet zu Mergesort)
Wie im Buch von Sedgewick ausgeführt, gestatten Rot-Schwarz-Bäume einen Top-Down-Ansatz zur Rebalancierung, was insbesondere für ihre parallele Ausführung erhebliche Vorteile bringt.