Gliederung des Kapitels

3.2 Suchverfahren

~~3.2.1 Suchen in sequentiellen Strukturen~~

~~3.2.2 Bäume und (binäre) Suchbäume~~

3.2.3 Optimale Suchbäume

3.2.4 Balancierte Bäume, AVL-Bäume

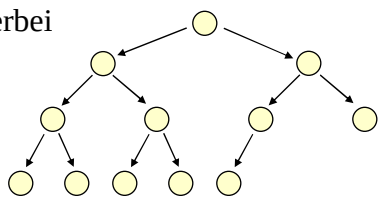
3.2.5 B-Bäume

3.2.6 Digitale Suchbäume (Tries)

3.2.3 Optimale Suchbäume

Eine geordnete Folge von Elementen $a_1, a_2, a_3, ..., a_n$ (mit $a_i \leq a_j$ für $i < j$) wird oft als Binärbaum gespeichert. Unter den C_n möglichen Binärbäumen wird man sich denjenigen auswählen, in dem man jedes Element möglichst schnell finden kann. Wird nach allen Elementen mit gleicher Wahrscheinlichkeit gesucht, so wird man einen möglichst "gleichförmigen" Baum nehmen, also einen Baum der unten stehenden Art.

Ausgehend von der Wurzel wird hierbei die nächste Knotenschicht von links aufgefüllt, bis n Knoten vorhanden sind. Alle Blätter haben hier das Level log(n+1), vgl. 3.2.2.12.



Wird dagegen nach jedem Element a_i mit der Wahrscheinlichkeit oder der Häufigkeit p_i gesucht, dann wird man *den* Binärbaum auswählen, für den die mittlere Suchdauer, gewichtet mit den Häufigkeiten, minimal ist (vgl. 3.2.2.13).

Definition 3.2.3.1: Gegeben sind eine geordnete Folge von n Elementen $a_1, a_2, a_3, ..., a_n$ (mit $a_i \leq a_j$ für $i < j$) mit zugehörigen Häufigkeiten oder Wahrscheinlichkeiten $p_1, p_2, p_3, ..., p_n$ (insbesondere sind alle $p_i \geq 0$) sowie ein Suchbaum B mit n Knoten $v_1, ..., v_n$, wobei a_i der Inhalt des Knotens v_i ist.

Die gewichtete mittlere Suchdauer von B ist definiert als

$$S(B) = \sum_{i=1}^{n} p_i \cdot \text{level}(v_i) .$$

Man kann in dieser Definition die Werte p_i als Häufigkeiten verwenden: Man führt p Anfragen durch und zählt hierbei, wie oft nach jedem Element a_i gefragt wurde; diese Anzahl sei jeweils p_i. Es gilt $p_1 + p_2 + p_3 + ... + p_n = p$.

Man kann in der Definition zum einen diese Werte p_i verwenden, man kann aber auch die relativen Häufigkeiten p_i/p benutzen; diese sind die Wahrscheinlichkeiten für das Auftreten einer Anfrage nach dem i-ten Element a_i. Der Begriff "mittlere Suchdauer" hat seine umgangssprachliche Bedeutung allerdings nur, wenn man Wahrscheinlichkeiten zugrunde legt.

Das folgende Lösungsverfahren arbeitet sowohl mit Häufigkeiten als auch mit Wahrscheinlichkeiten.

Hinweis: Die in 3.2.2.13 bereits definierte mittlere Suchzeit MS(B) ist die gewichtete mittlere Suchdauer für den Fall, dass alle Elemente die gleiche Wahrscheinlichkeit $p_i = 1/n$ besitzen.

Hat man die Elemente $a_1, a_2, a_3, ..., a_n$ in einem Suchbaum B abgelegt und wird mit den Wahrscheinlichkeiten p_i nach ihnen gesucht, so gibt S(B) die zu erwartende Anzahl der Vergleiche an, um ein beliebiges Element zu finden.

Hierbei suchen wir zunächst nur nach Elementen a_i, die in der ursprünglichen Folge $a_1, a_2, a_3, ..., a_n$ vorkommen. Wird auch nach Elementen gesucht, die nicht zu den a_i gehören, so muss man die Bäume leicht abändern; dies wird am Ende dieses Abschnitts im Hinweis 4 erläutert.

Definition 3.2.3.2: Gegeben sei eine geordnete Folge von n Elementen $a_1, a_2, a_3, ..., a_n$ (mit $a_i \leq a_j$ für $i < j$) mit zugehörigen Wahrscheinlichkeiten $p_1, p_2, p_3, ..., p_n$ ($p_i \geq 0$ und $p_1+p_2+...+p_n = 1$) oder Häufigkeiten $p_1, p_2, p_3, ..., p_n$ ($p_i \geq 0$). Ein Suchbaum B mit n Knoten $v_1, ..., v_n$, wobei a_i der Inhalt des Knotens v_i ist, heißt optimaler Suchbaum, wenn für alle anderen Suchbäume B' mit n Knoten $v'_1, ..., v'_n$ (wobei a_i der Inhalt des Knotens v'_i ist) gilt: $S(B) \leq S(B')$.

In diesen Definitionen sind nicht die konkreten Elemente a_i, sondern nur deren Wahrscheinlichkeiten/Häufigkeiten p_i von Bedeutung.
Die Aufgabe lautet nun, zu $p_1, p_2, p_3, ..., p_n$ einen optimalen Suchbaum zu konstruieren. Da es C_n mögliche Suchbäume gibt (vgl. Satz 3.2.2.7), dauert das systematische Durchprobieren viel zu lange.

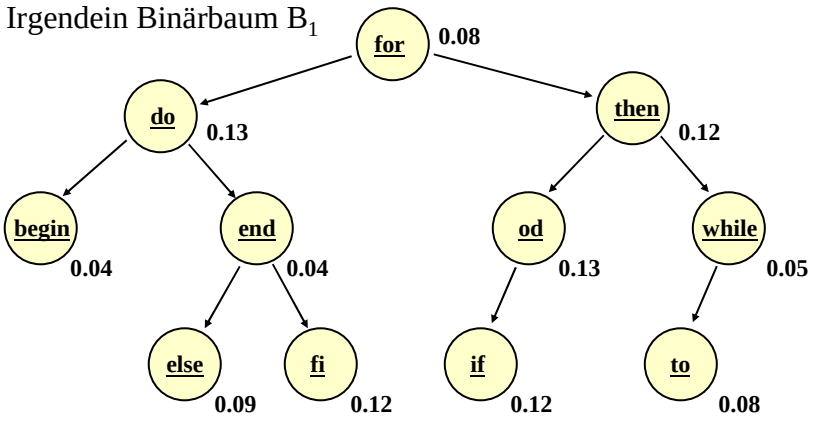
3.2.3.3 Beispiel

Als Beispiel betrachten wir einen Compiler, der ein Programm übersetzen soll. Hierzu muss er zunächst die Schlüsselwörter der Sprache erkennen. Diese Wörter treten mit gewissen Wahrscheinlichkeiten in einem Programm auf und wir nehmen an, wir hätten diese Wahrscheinlichkeiten gemessen. Wir verwenden hier die n=11 Wörter begin, do, else, end, fi, for, if, od, then, to, while. Ihre Wahrscheinlichkeiten seien

begin: 0.04, do: 0.13, else: 0.09, end: 0.04, fi: 0.12, for: 0.08, if: 0.12, od: 0.13, then: 0.12, to: 0.08, while: 0.05.

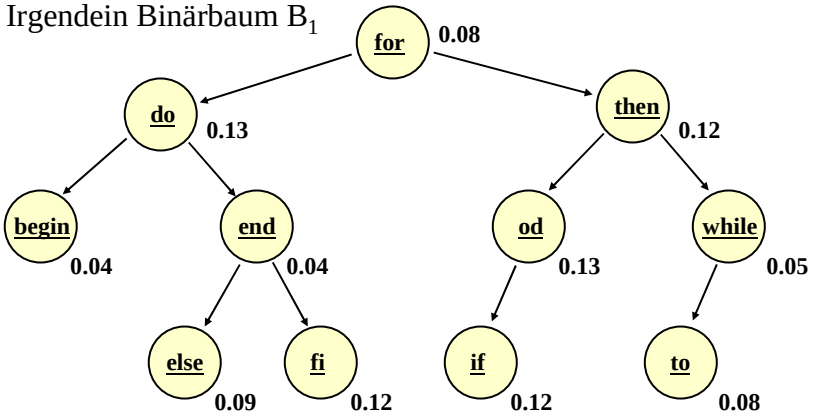
Wir untersuchen zunächst einen beliebigen Binärbaum B_1 für diese 11 Wörter und berechnen dessen gewichtete mittlere Suchdauer $S(B_1)$.

Irgendein Binärbaum B_1



Schlüsselwörter eintragen (inorder-Durchlauf)
Wahrscheinlichkeiten hinzufügen
Gewichtete mittlere Suchdauer nun ausrechnen (bitte selbst durchführen, dann erst zur nächsten Folie klicken).

Irgendein Binärbaum B_1

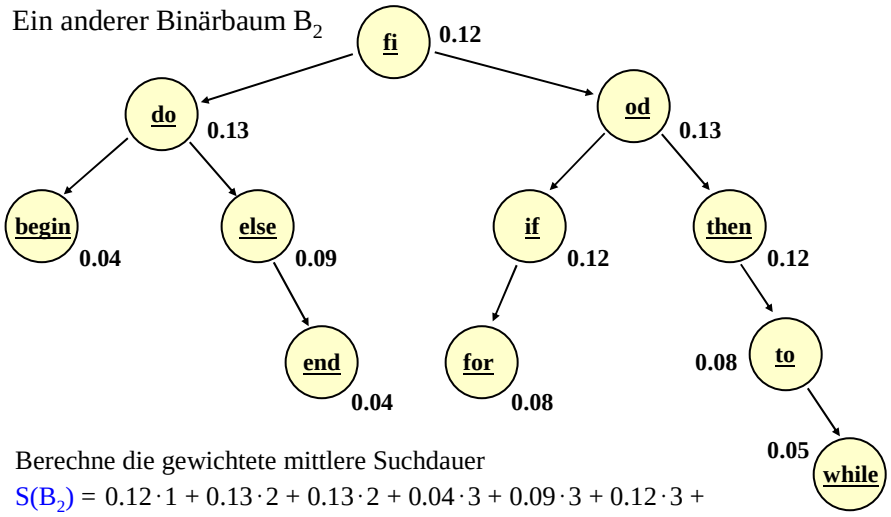


Berechne die gewichtete mittlere Suchdauer

$S(B_1) = 0.08 \cdot 1 + 0.13 \cdot 2 + 0.12 \cdot 2 + 0.04 \cdot 3 + 0.04 \cdot 3 + 0.13 \cdot 3 + 0.05 \cdot 3 + 0.09 \cdot 4 + 0.12 \cdot 4 + 0.12 \cdot 4 + 0.08 \cdot 4 = 3.00$

Konstruieren Sie nun einen Suchbaum mit kleinerem S(B). [Man sieht sofort, dass man to ein Level hinauf und while eines hinab schieben sollte, usw.]

Ein anderer Binärbaum B_2



Berechne die gewichtete mittlere Suchdauer

$S(B_2) = 0.12 \cdot 1 + 0.13 \cdot 2 + 0.13 \cdot 2 + 0.04 \cdot 3 + 0.09 \cdot 3 + 0.12 \cdot 3 + 0.12 \cdot 3 + 0.04 \cdot 4 + 0.08 \cdot 4 + 0.08 \cdot 4 + 0.05 \cdot 5 = 2.80$

Konstruieren Sie nun weitere Suchbäume mit kleinerem S(B). Versuchen Sie, den optimalen Suchbaum aufzuspüren. Erkennen Sie hierbei ein Verfahren? ■

Hilfssatz 3.2.3.4:

Jeder Unterbaum eines optimalen Suchbaums ist ebenfalls ein optimaler Suchbaum.

Beweis: Wäre der Unterbaum U eines optimalen Suchbaums B nicht optimal, so gäbe es einen anderen Suchbaum U', der bzgl. der in U enthaltenen Elemente optimal ist, für den insbesondere S(U') < S(U) gilt. Tausche dann im Baum B den Unterbaum U gegen den Unterbaum U' aus, wodurch der Baum B' entsteht. Es gilt dann ($a_i \in$ U bezeichnen die Elemente, die im Unterbaum U liegen; x+1 sei das Level der Wurzel von U in B; in U und U' liegen natürlich die gleichen Elemente):

$$S(B) = \sum_{i=1}^{n} p_i \cdot level(v_i) = \sum_{a_i \in B-U} p_i \cdot level(v_i) + \sum_{a_i \in U} p_i \cdot level(v_i)$$

$$S(B) = \sum_{a_i \in B-U} p_i \cdot level(v_i) + \overbrace{\sum_{a_i \in U} p_i \cdot (level(v_i) - x)}^{S(U)} + \sum_{a_i \in U} p_i \cdot x$$

Hierbei ist x+1 das Level der Wurzel des Unterbaums U in B; dies ist zugleich das Level der Wurzel des Unterbaums U' in B'.

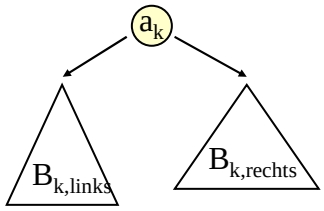
$$S(B) = \sum_{a_i \in B-U} p_i \cdot level(v_i) + S(U) + \sum_{a_i \in U} p_i \cdot x$$

$$> \sum_{a_i \in B'-U'} p_i \cdot level(v'_i) + S(U') + \sum_{a_i \in U'} p_i \cdot x$$

$$= \sum_{a_i \in B'-U'} p_i \cdot level(v'_i) + \sum_{a_i \in U'} p_i \cdot (level(v'_i) - x) + \sum_{a_i \in U'} p_i \cdot x = S(B')$$

Also war B kein optimaler Suchbaum, im Widerspruch zur Voraussetzung. Folglich muss U optimal gewesen sein. ■

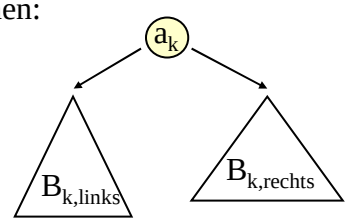
Hieraus folgt, dass man einen optimalen Suchbaum schrittweise aus seinen Unterbäumen aufbauen kann.
Um den optimalen Suchbaum für $a_i, a_{i+1}, ..., a_{j-1}, a_j$ zu finden, prüft man alle Paare von Unterbäumen für $a_i, a_{i+1}, ..., a_{k-1}$ und $a_{k+1}, a_{i+2}, ..., a_j$ durch und wählt die Kombination aus, deren Summe den kleinsten Wert ergibt. Skizze hierzu:



Der beste Fall liegt vor, wenn $S(B_{k,links}) + S(B_{k,rechts})$ minimal ist. Ein solches k ist also zwischen i und j zu suchen.

Dieser Baum enthält genau die Elemente $a_i, a_{i+1}, ..., a_{j-1}, a_j$; es ist $i \le k \le j$.

3.2.3.5 Bezeichnungen und Formeln: Gegeben seien n und die Häufigkeiten P[1], P[2], ..., P[n]. Sei $1 \le i \le j \le n$.
Es sei G[i,j] = P[i] + P[i+1] + ... + P[j] (man bezeichnet diesen Wert auch als das "Gewicht" der Teilfolge a_i bis a_j).
Mit S[i,j] bezeichnen wir die gewichtete mittlere Suchdauer für einen optimalen Suchbaum für die Elemente $a_i, a_{i+1}, ..., a_{j-1}, a_j$.
S[i,j] lässt sich leicht rekursiv berechnen:
S[i,i] = P[i] für i = 1, 2, ..., n.



Die Formel für S[i,j] liest man leicht aus der nebenstehenden Skizze ab.
S[i,j] ist S[i,k-1] + S[k+1,j] plus der Erhöhung aller Level um 1 (dieser Summand ist genau G[i,j]):

$S[i,j] = \text{Min}\{ S[i,k-1] + S[k+1,j] \mid i \le k \le j \} + G[i,j]$ für i < j.

Den Wert S[i,j] kann man berechnen, wenn man alle Werte S[r,s] mit s-r < j-i kennt. Setze also diff:=j-i und berechne für diff=0,1,2,...,n-1 alle Werte S[i,i+diff].

Für diff=0 setze S[i,i] =P[i]. Für diff > 0 verwende die Rekursionsformel für S[i,j] mit j=i+diff.

Zeitaufwand: Für das eigentliche Verfahren benötigen drei ineinander geschachtelte Schleifen:

```
for diff in 1..n-1 loop
    for i in 1..n-diff loop
        j := i + diff;
        for k in i..j loop berechne das Minimum aller Werte
            S[i,k-1] + S[k+1,j]; end loop; setze S[i,j] entsprechend;
    end loop;
end loop;
```

Das gesuchte Ergebnis S(B) steht am Ende in S[1,n].

Da alle Schleifen von n abhängen, beträgt der Aufwand $\Theta(n^3)$.

Erläuterungen zum Programm:

Wir verwenden G[i,j] und S[i,j] wie angegeben, wobei wir rund 50% des Speicherplatzes verschwenden, da wir diese Werte nur für $i \leq j$ brauchen.

Weiterhin berechnen wir in diesem Programm auch die Wurzeln der Unterbäume zu a_i bis a_j. Diese legen wir in einem Feld R ab, wobei gilt

R[i,j] = k, so dass a_k in der Wurzel des optimalen Suchbaums von a_i, a_{i+1}, ..., a_j steht. (Dieses k wird im Programm in der innersten Schleife als k_min berechnet.)

Mit den R[i,j] kann man am Ende den optimalen Suchbaum konstruieren (dies programmieren wir aber nicht aus).

Die Minimumbildung erfolgt wie üblich, indem man alle Werte durchprobiert und das aktuelle Minimum speichert.

Die Werte G[i,j] kann man zu Beginn des Programms berechnen; wir führen dies jedoch in der mittleren Schleife durch.

3.2.3.6: Programm für einen optimalen Suchbaum in Zeit $\Theta(n^3)$

Global gegeben seien: Die Zahl n und n Wahrscheinlichkeiten P[1], ..., P[n].
Ergebnis ist der Wert S[1,n] = die Suchdauer eines optimalen Suchbaums zu diesen Wahrscheinlichkeiten. Aus den Wurzeln R[i,j] der Teilbäume kann man den optimalen Suchbaum rekonstruieren. (Wie? Selbst hinzu programmieren!)

```
var  i, j, k, k_min, diff: natural;  min: real;
     S, G: array [1..n+1, 1..n] of real; R: array [1..n, 1..n] of natural;
begin  for i:=1 to n do
          S[i+1,i] := 0.0; S[i,i] := P[i]; G[i,i] := P[i]; R[i,i]:=i od;
       for diff:=1 to n-1 do
          for i:=1 to n-diff do
             j := i + diff; G[i,j] := G[i,j-1] + P[j]; min := S[i+1,j]; k_min := i;
             for k:=i+1 to j do
                if  S[i,k-1] + S[k+1,j] < min  then
                   min := S[i,k-1] + S[k+1,j]; k_min := k  fi  od;
          S[i,j] := min + G[i,j]; R[i,j] := k_min
       od od   -- Die gewichtete mittlere Suchdauer eines optimalen Suchbaums steht nun
end             -- in R[1,n]; der Suchbaum selbst kann aus den R[i,j] konstruiert werden.
```


3.2.3.7: Hinweise

Hinweis 1: Dieses ist ein "garantiertes" n^3-Verfahren. In der Praxis ist es daher nur für kleinere Werte von n einsetzbar.
Hinweis 2: Das Suchen (FIND) lässt sich optimal schnell durchführen. Dagegen muss man beim Einfügen (INSERT) und beim Löschen (DELETE) den Baum neu aufbauen. Daher setzt man optimale Suchbäume nur dann ein, wenn der Datenbestand sich über längere Zeiträume nicht ändert. Beispiele hierfür sind Lexika oder die Erkennung von Schlüsselwörtern und anderen Textteilen durch einen Compiler.
In zeitkritischen Anwendungen kann man eine Doppel-Strategie verfolgen: Der "große Datenbestand" ist in einem optimalen Suchbaum gespeichert, die (seltenen) Neueintragungen speichert man in einem gesonderten Binärbaum, bis dieser eine gewisse Größe erreicht hat; dann baut man aus den beiden Bäumen einen neuen optimalen Suchbaum auf.

Hinweis 3: Man kann sich jedoch überlegen, dass die Wurzel des jeweils zu konstruierenden Unterbaums nur innerhalb bestimmter Grenzen liegen kann, die von den im letzten Durchlauf konstruierten Wurzeln bestimmt wird, genauer: Es gilt stets $R[i,j-1] \leq R[i,j] \leq R[i+1,j]$. Dies nennt man die "Monotonie der Wurzeln".
Den Beweis finden Sie in Lehrbüchern oder in weiterführenden Vorlesungen (Theorie III oder Effiziente Algorithmen).
Mit dieser Eigenschaft lässt sich obiges Verfahren zu einem $\Theta(n^2)$-Verfahren beschleunigen. (Selbst durchdenken. Obiges Programm muss nur leicht modifiziert werden.)
Aber auch diese Beschleunigung reicht für die Praxis nicht aus, wenn sich der Datenbestand und/oder die Wahrscheinlichkeiten häufig ändern. Man verwendet dann AVL-Bäume oder B-Bäume (siehe 3.2.4 und 3.2.5).

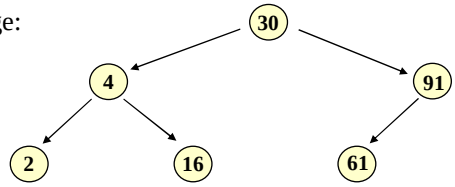
Hinweis 4: Zum Abschluss erläutern wir kurz, wie man dieses Verfahren auf den Fall der erfolglosen Suche erweitert.

Sucht man nach einem Element, das nicht in der Folge $a_1, a_2, a_3, ..., a_n$ vorkommt, so endet die Suche bei einem der n+1 null-Zeiger des Suchbaums.

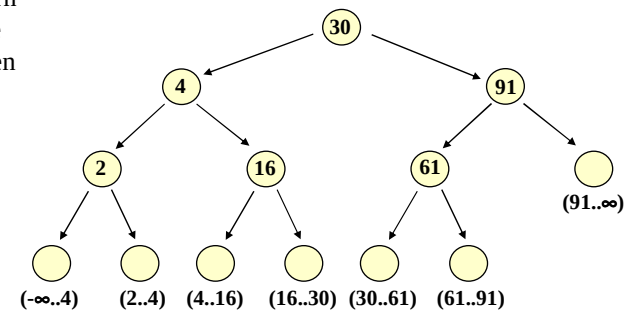
Man ersetzt nun diese null-Zeiger durch Knoten, an denen die Suche endet und deren Inhalt alle die Elemente bilden, mit denen man im Suchbaum hierhin gelangt. Ihre Inhalte sind daher offene Intervalle der Form $(i..j) = \{k \mid i < k < j\}$.

Jetzt muss man den Intervallen ebenfalls Wahrscheinlichkeiten zuordnen, mit denen nach einem Element in ihnen gesucht wird. Auf diese Weise kann man einen optimalen Suchbaum auch für den Fall der "erfolglosen Suche" mit dem gleichen Verfahren konstruieren.

Beispiel: Gegebene Folge: 2, 4, 16, 30, 61, 91 und ein zugehöriger Suchbaum.



Füge an den Blättern neue Blätter an, die die nicht enthaltenen Elemente repräsentieren; diese neuen Blätter werden mit den offenen Intervallen (i..j) beschriftet.



3.2.4 Balancierte Bäume, AVL-Bäume

Unter der Balance eines Knotens in einem Binärbaum versteht man das Verhältnis, in dem seine beiden Unterbäume zueinander stehen.

Es sei u ein Knoten und es seien UB_{links} und UB_{rechts} seine beiden Unterbäume. Es bezeichnen $|V_{UB_{links}}|$ die *Anzahl der Knoten* im linken Unterbaum und $|V_{UB_{rechts}}|$ die *Anzahl der Knoten* im rechten Unterbaum von u. $|V_{UB_{links}}| + |V_{UB_{rechts}}|$ ist dann die Anzahl aller Knoten, die sich unterhalb des Knotens u befinden. Es sei T die Tiefe eines Baums (3.2.2.1,11).

Definition 3.2.4.1:

Es sei α eine Zahl aus dem reellen Intervall $(0, ½]$. Ein Knoten u eines Binärbaums heißt α-gewichtsbalanciert, wenn gilt

$$\alpha \leq \frac{|V_{UB_{links}}| + 1}{|V_{UB_{links}}| + |V_{UB_{rechts}}| + 2} \leq 1 - \alpha$$

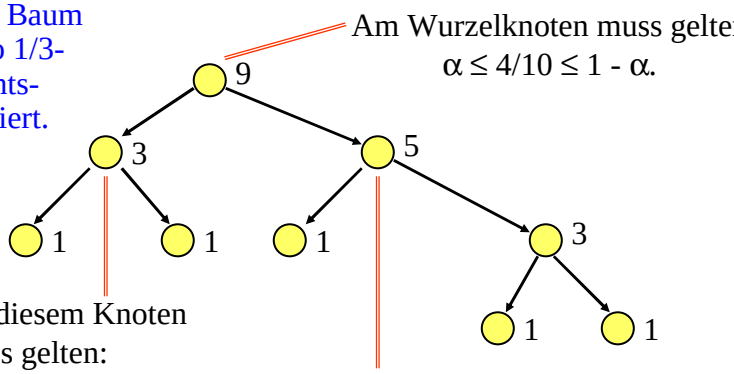
Den Ausdruck $\beta(u) = \frac{|V_{UB_{links}}| + 1}{|V_{UB_{links}}| + |V_{UB_{rechts}}| + 2}$

nennt man auch die Gewichts- oder Wurzelbalance von u.

Die Anzahlen der Knoten in den Unterbäumen werden hier jeweils um 1 erhöht, weil die Unterbäume leer sein können.

Beispiel 3.2.4.2: An jedem Knoten sei die Zahl der Knoten in dem Unterbaum, dessen Wurzel er ist, angegeben. Bestimme für folgenden Baum ein geeignetes α.

Dieser Baum ist also 1/3-gewichts-balanciert.



Folgerung 3.2.4.3: Symmetrie der Balance

Wenn

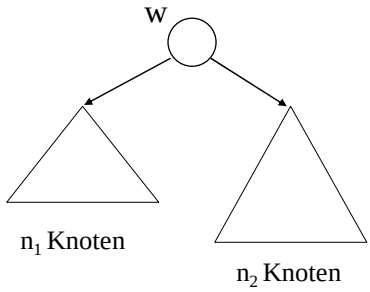
$$\alpha \leq \frac{|V_{UB_{links}}| + 1}{|V_{UB_{links}}| + |V_{UB_{rechts}}| + 2} \leq 1 - \alpha$$

gilt, dann gilt auch

$$\alpha \leq \frac{|V_{UB_{rechts}}| + 1}{|V_{UB_{links}}| + |V_{UB_{rechts}}| + 2} \leq 1 - \alpha$$

Der Beweis ist einfach: Seien $|V_{UB_{links}}| = n_1$ und $|V_{UB_{rechts}}| = n_2$, dann ist die Gesamtzahl der Knoten $n = n_1 + n_2 + 1$. Die Aussage folgt sofort aus $\alpha \leq (n_1+1)/(n+1) = 1 - (n_2+1)/(n+1) \leq 1 - \alpha$. ■

Betrachte einen Binärbaum B mit n Knoten und der Wurzel w:



Die Gesamtzahl der Knoten im Baum ist $n = n_1 + n_2 + 1$.

Für α-gewichtsbalancierte Bäume B gilt dann an der Wurzel w für i = 1, 2:

$$\alpha \leq \frac{n_i + 1}{n + 1} \leq 1 - \alpha, \text{ d.h., } n_i \leq (1 - \alpha)\cdot(n + 1) - 1 = (1 - \alpha)\cdot n - \alpha$$

Dies gilt auch für das nächste Level, d.h.:

Anzahl der Knoten in einem Unterbaum zwei Level tiefer

$\leq (1 - \alpha) \cdot n_i - \alpha$

$\leq (1 - \alpha) \cdot ((1 - \alpha) \cdot n - \alpha) - \alpha = (1 - \alpha)^2 \cdot n - \alpha \cdot ((1 - \alpha) + 1)$

Analog weiter einsetzen! Dies ergibt (beachte $\alpha > 0$):
Anzahl der Knoten in einem Unterbaum k Level tiefer

$\leq (1-\alpha)^k \cdot n - \alpha \cdot ((1-\alpha)^{k-1} + (1-\alpha)^{k-2} + (1-\alpha)^1 + (1-\alpha)^0)$

$= (1-\alpha)^k \cdot n - \alpha \cdot (1 - (1 - \alpha)^k) / (1 - (1 - \alpha))$

$= (1-\alpha)^k \cdot n - (1 - (1 - \alpha)^k)$

$< (1-\alpha)^k \cdot n$

Wenn $(1-\alpha)^k \cdot n < 2$ geworden ist, gibt es höchstens noch einen Knoten und dieser muss dann ein Blatt sein.

Aus $(1-\alpha)^k \cdot n < 2$ folgt mit $z := 1/(1 - \alpha)$:

$n < 2 \cdot z^k$, d.h., $\log(n/2) < k \cdot \log(z) = - k \cdot \log(1-\alpha)$

Suche das kleinste k, für das diese Ungleichung zutrifft:
$k = 1 - (\log(n) - 1) / \log(1-\alpha) \in O(\log(n))$.

Dieses (reelle) k ist eine obere Schranke für die Tiefe des Baums (minus 1). Somit haben wir gezeigt:

Satz 3.2.4.4
Die Tiefe eines α-gewichtsbalancierten Baums mit n Knoten ist höchstens $2 - (\log(n) - 1) / \log(1-\alpha)$,
d.h., die Tiefe ist von der Größenordnung $O(\log(n))$.
Je mehr α sich der Zahl 0.5 nähert, um so mehr nähert sich die Tiefe dem minimalen Wert $\lceil \log(n+1) \rceil$ (vgl. diskreter Zweierlogarithmus 3.2.2.11 und anschließende Folgerung).

Kann man die Operationen FIND, INSERT und DELETE auf solchen α-gewichtsbalancierten Bäumen so ausführen, dass diese Operationen schnell durchführbar sind (z.B. in O(log(n)) Schritten) und dass nach Ausführung jeder Operation der entstandene Baum wieder ein α-gewichtsbalancierter Baum ist?

Ja, das geht tatsächlich.

Wir führen dies hier nicht durch, sondern verweisen auf die Literatur.

An Stelle der gewichtsbalancierten Bäume behandeln wir die höhenbalancierten Bäume, die besonders angenehme Eigenschaften haben.

Definition 3.2.4.5:

Ein binärer Baum heißt **AVL-Baum** (oder höhenbalancierter Baum), wenn für jeden Knoten u gilt:

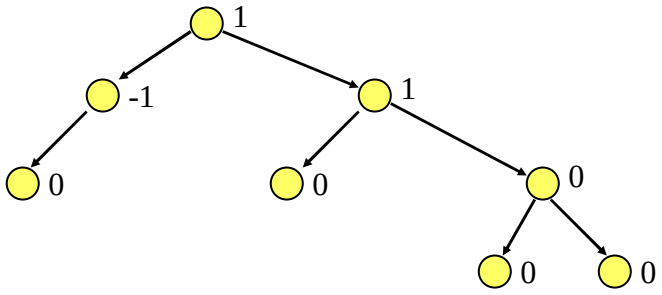
$T(UB_{rechts}) - T(UB_{links}) =$
("Höhe" des rechten Unterbaums von u -
"Höhe" des linken Unterbaums von u) $\in \{-1, 0, 1\}$,

d.h., für jeden Knoten unterscheiden sich die Höhen (= Tiefen) seiner Unterbäume höchstens um 1.

$T(UB_{rechts}) - T(UB_{links})$ heißt auch der Balancefaktor von u.

AVL-Bäume wurden benannt nach ihren beiden Erfindern Adelson-Velski und Landis.

Beispiel: Der bereits im Beispiel 3.2.4.2 betrachtete Baum ist höhenbalanciert, d.h. ein AVL-Baum. Wir fügen die Balancefaktoren neben jedem Knoten an:



Beispiel: Dagegen ist folgender Baum *nicht* höhenbalanciert: