Programm in Python

```
def delete(self) :
    if self.left == None and self.right == None :
        if self.parent.left == self :
            self.parent.left = None
        else :
            self.parent.right = None
    elif self.left == None :
        self.right.parent = self.parent
        if self.parent.left == self :
            self.parent.left = self.right
        else :
            self.parent.right = self.right
    else :
        max = self.left
        while max.right :
            max = max.right
        self.key, self.value = max.key, max.value
        max.delete()
```

Binäre Suchbäume – Löschen

In der Klasse Searchtree⟨K, D⟩:

Java

```
public void delete(K k) {
  if(root == null) return;
  if(root.left == null && root.right == null && root.key == k)
    root = null;
  else {
    Searchtreenode⟨K, D⟩ n = root.findsubtree(k);
    if(n ≠ null) n.delete();
  }
}
```

Binäre Suchbäume – Löschen

Java

```
void delete() {
  if(left == null && right == null) {
    if(parent.left == this) parent.left = null;
    else parent.right = null; }
  else if(left == null) {
    if(parent.left == this) parent.left = right;
    else parent.right = right;
    right.parent = parent; }
  else {
    Searchtreenode⟨K, D⟩ max = left;
    while(max.right ≠ null) max = max.right;
    copy(max); max.delete();
  }
}
```

Binäre Suchbäume – Löschen

In Searchtree⟨K, D⟩ jetzt korrekt:

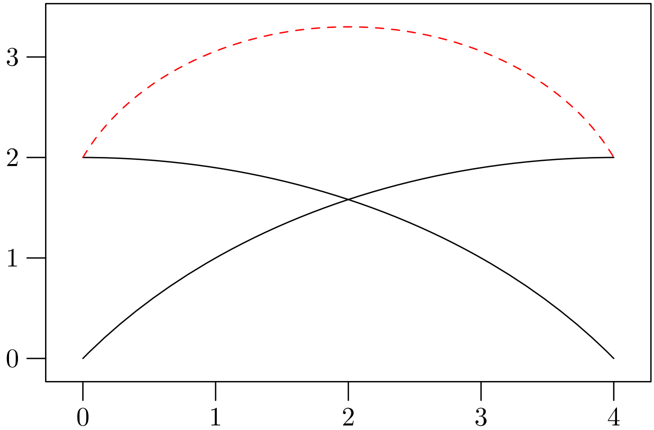
Java

```
public void delete(K k) {
  if(root == null) return;
  if(root.key.equals(k))
    if(root.left == null && root.right == null) {
      root = null;  return;
    }
    else if(root.left == null) {
      root = root.right;  root.parent = null;  return;
    }
  Searchtreenode⟨K, D⟩ n = root.findsubtree(k);
  if(n ≠ null) n.delete();
}
```

Binäre Suchbäume – Beispiel

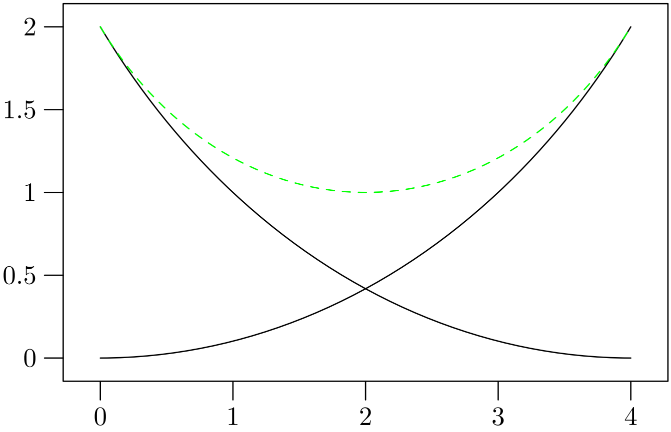
Die Schlüssel von 1 bis 40 werden zufällig eingefügt oder gelöscht.

Binäre Suchbäume – Analyse



Summe schlechte Näherung des Maximums.

Binäre Suchbäume – Analyse



Summe gute Näherung des Maximums.

Die Kurven sind steiler.

Binäre Suchbäume – Analyse

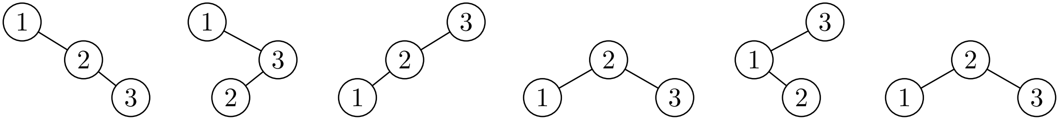
Wir fügen die Knoten $1, \ldots, n$ in zufälliger Reihenfolge in einen leeren Suchbaum ein.

Sei T_n die Höhe dieses Suchbaums.

Wir interessieren uns für $E(T_n)$.

Wir betrachten erst einmal T_0, T_1, T_2 und T_3:

- $E(T_0) = 0$
- $E(T_1) = 1$
- $E(T_2) = 2$
- $E(T_3) = 8/3$



Allgemeiner Fall:

Die Wurzel des Baums enthält $W \in \{1, \ldots, n\}$.

$$\Pr[W = k] = 1/n \text{ für } k \in \{1, \ldots, n\}$$

Wie sieht der Rest des Baums aus, falls $W = k$?

In den linken Teilbaum wurden $\{1, \ldots, k-1\}$ in zufälliger Reihenfolge eingefügt.

Seine Höhe ist T'_{k-1}.

Die Höhe des rechten Teilbaums ist T''_{n-k}.
Die Gesamthöhe ist $T_n = \max\{T'_{k-1}, T''_{n-k}\} + 1$.

Die Gesamthöhe ist $T_n = \max\{T'_{k-1}, T''_{n-k}\} + 1$.

$$E(T_n) = \frac{1}{n} \sum_{k=1}^{n} E(\max\{T'_{k-1}, T''_{n-k}\} + 1)$$

Wir können das Maximum durch die Summe abschätzen:

$$E(T_n) \leq \frac{1}{n} \sum_{k=1}^{n} \Bigl(E(T_{k-1}) + E(T_{n-k}) + 1\Bigr)$$

Leider zu grob! Führt zu einer schlechten Abschätzung.

Problem:

$E(\max\{X, Y\}) \le E(X) + E(Y)$ korrekt, aber zu ungenau.

$E(\max\{X, Y\}) \le \max\{E(X), E(Y)\}$ genau genug, aber zu unkorrekt.

Führe neue Zufallsvariablen ein:

$$\hat{T}_n = 2^{T_n}, \ \hat{T}'_n = 2^{T'_n}, \ \hat{T}''_n = 2^{T''_n}$$

$$ET_n = \frac{1}{n}\sum_{k=0}^{n-1} E\Bigl(\max\{T'_k, T''_{n-k-1}\} + 1\Bigr)$$

$$E\hat{T}_n = \frac{1}{n}\sum_{k=0}^{n-1} E\Bigl(2^{\max\{T'_k, T''_{n-k-1}\}+1}\Bigr)$$

Die Kurven sind jetzt steiler!

Vereinfachen wir diese Rekursionsgleichung zunächst:

$$
\begin{aligned}
E\,\hat{T}_n = \frac{1}{n}\sum_{k=0}^{n-1} E\left(2^{\max\{T'_k, T''_{n-k-1}\}+1}\right) = \\
= \frac{1}{n}\sum_{k=0}^{n-1} 2E\left(\max\{2^{T'_k}, 2^{T''_{n-k-1}}\}\right) = \frac{2}{n}\sum_{k=0}^{n-1} E\left(\max\{\hat{T}'_k, \hat{T}''_{n-k-1}\}\right) \\
\le \frac{2}{n}\sum_{k=0}^{n-1} E\left(\hat{T}'_k + \hat{T}''_{n-k-1}\}\right) = \frac{2}{n}\sum_{k=0}^{n-1}\left(E\,\hat{T}_k + E\,\hat{T}_k\}\right) = \frac{4}{n}\sum_{k=0}^{n-1} E\,\hat{T}_k
\end{aligned}
$$

Diese Rekursionsgleichung läßt sich mit Standardmethoden lösen.
→ Vorlesung Analyse von Algorithmen

Wir müssen sie aber gar nicht exakt lösen.
Ähnliche Situation bei der Analyse von Quicksort (später).

$$E\hat{T}_n = \frac{4}{n}\sum_{k=0}^{n-1} E\hat{T}_k$$

Wir „lösen“ diese Gleichung nicht.

Wir zeigen nur, daß $E\hat{T}_n \leq (n+3)^{\underline{3}} = (n+3)(n+2)(n+1)$:

$n = 1$: $E\hat{T}_1 = 2 \leq (1+3)^{\underline{3}}$

$n > 1$:

$$E\hat{T}_n \leq \frac{4}{n}\sum_{k=0}^{n-1} E\hat{T}_k \overset{\text{I.V.}}{\leq} \frac{4}{n}\sum_{k=0}^{n-1}(k+3)^{\underline{3}} = \frac{4}{n}\frac{(n+3)^{\underline{4}}}{4} = (n+3)^{\underline{3}}.$$

Polynome verhalten sich gut beim Integrieren.

Lemma

$$\int_0^x t^k \, dt = \frac{x^{k+1}}{k+1}$$

$$\sum_{i=0}^{n-1} i^{\underline{k}} = \frac{n^{\underline{k+1}}}{k+1}$$

Fallende Potenzen verhalten sich gut beim Summieren.

Lemma

Es seien $p_1, \ldots, p_n \in [0,1]$ *mit* $p_1 + \ldots + p_n = 1$.

Des weiteren seien $w_1, \ldots, w_n \geq 0$.

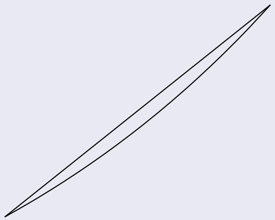
Dann gilt

$$2^{\sum_{k=1}^{n} w_k p_k} \leq \sum_{k=1}^{n} 2^{w_k} p_k.$$

Beweis.

Wir betrachten $f(t) = 2^a t + 2^b(1-t)$ und $g(t) = 2^{at+b(1-t)}$ im Einheitsintervall. Es sei $a \neq b$.

- f ist linear
- g ist überall links- oder rechtsgekrümmt: Denn $g''(t) = g(t)(\ln 2)^2(a-b)$.
- O.B.d.A. $a > b$, dann ist g linksgekrümmt.
- $f(0) = g(0) \leq f(1) = g(1) \Rightarrow f(t) \geq g(t)$



Daraus folgt

$$2^{\sum_{k=1}^n w_k p_k} \leq \sum_{k=1}^n 2^{w_k} p_k$$

für $n = 2$. □

Beweis.

Für $n > 2$ gilt:

$$2^{\sum_{k=1}^{n} kp_k} = 2^{np_n+\sum_{k=1}^{n-1} kp_k} = 2^{np_n+\left(\sum_{k=1}^{n-1}\frac{kp_k}{1-p_n}\right)(1-p_n)} \leq$$

$$\overset{\text{I.V.}}{\leq} 2^n p_n + 2^{\sum_{k=1}^{n-1}\frac{kp_k}{1-p_n}}(1-p_n) \leq$$

$$\overset{\text{I.V.}}{\leq} 2^n p_n + \left(\sum_{k=1}^{n-1} 2^k \frac{p_k}{1-p_n}\right)(1-p_n) = \sum_{k=1}^{n} 2^k p_k$$

□

Kommen wir zurück zu

$$E\hat{T}_n \leq (n+3)^{\underline{3}}.$$

Es gilt:

$$2^{ET_n} = 2^{\sum_{k=1}^{n} k \Pr[T_n=k]} \leq \sum_{k=1}^{n} 2^k \Pr[T_n = k] = E(2^{T_n})$$

Und damit:

$$ET_n \leq \log(E\hat{T}_n) \leq \log((n+3)^{\underline{3}}) =$$

$$= \log(n^3(1 + O(1/n)) = 3\log(n) + O(1/n)$$

Fertig!

Theorem (Mittlere Höhe eines Suchbaums)

Werden in einen leeren binären Suchbaum n verschiedene Schlüssel in zufälliger Reihenfolge eingefügt, dann ist die erwartete Höhe des entstehenden Suchbaums $O(\log n)$.

Wir verzichten auf eine Analyse für gemischtes Einfügen und Löschen.

Hier ist nicht so viel bekannt.

Experimente zeigen gutes Verhalten.

Die Höhe eines Suchbaums ist für seine Geschwindigkeit maßgebend.

Theorem

Die Operationen Einfügen, Löschen und Suchen benötigen $O(h)$ Zeit bei einem binären Suchbaum der Höhe h.