5.2 Suchbäume

Eine gute Lösung für das Wörterbuchproblem besteht in der Verwendung von *Suchbäumen*. Ein binärer Baum wird im Speicher, ähnlich einer verketteten Liste, so abgelegt, dass jeder Knoten, zusätzlich zu seinem eigentlichen Inhalt dem Datensatz, noch zwei Zeiger enthält: einen auf das linke Kind und einen auf das rechte Kind. Wir schreiben dafür $v.D$, $v.links$ und $v.rechts$. Für Blätter werden diese beiden Zeiger auf NIL gesetzt.

Definition 5.1. Ein Suchbaum ist ein Baum der die folgende *Suchbaumeigenschaft* hat: für alle Knoten v gilt:

1. Für jeden Knoten w im linken Teilbaum von v gilt: $w.D.x < v.D.x$.

2. Für jeden Knoten w im rechten Teilbaum von v gilt: $w.D.x > v.D.x$.

Die Suche in einem Suchbaum funktioniert dann wie in Algorithmus 12 beschrieben. Man be-

Algorithmus 12 Suche in einem Suchbaum

```
Prozedur SUCHE(v, x)
    Falls v = NIL dann
        Antworte "nicht gefunden"
    sonst falls v.D.x = x dann
        Antworte v.D
    sonst falls v.D.x < x dann
        Antworte SUCHE(v.rechts, x)
    sonst                                          ▷ v.D.x > x
        Antworte SUCHE(v.links, x)
    Ende Falls
Ende Prozedur
```

achte die Ähnlichkeit dieses Algorithmus zur binären Suche in einem sortierten Datenfeld. Algorithmus 12 hat Laufzeitkomplexität $O(h)$ wobei h die Höhe, d.h. die Länge des längsten Pfades, des Baums ist.

Die Liste aller Elemente die der Suchbaum enthält kann in aufsteigender Reihenfolge ausgegeben werden, indem er in *symmetrischer Reihenfolge* (engl. *inorder*) durchlaufen wird, also zuerst der linke Teilbaum, dann der aktuelle Knoten, dann der rechte Teilbaum, siehe Algorithmus 13.

Algorithmus 13 Durchlaufen eines Suchbaums in symmetrischer Reihenfolge

```
Prozedur SBAUSGABE(v)
    Falls v ≠ NIL dann
        SBAUSGABE(v.links)
        AUSGABE(v.D)
        SBAUSGABE(v.rechts)
    Ende Falls
Ende Prozedur
```

Ein einfacher Algorithmus zum Einfügen eines Elements D ist in Algorithmus 14 angegeben.

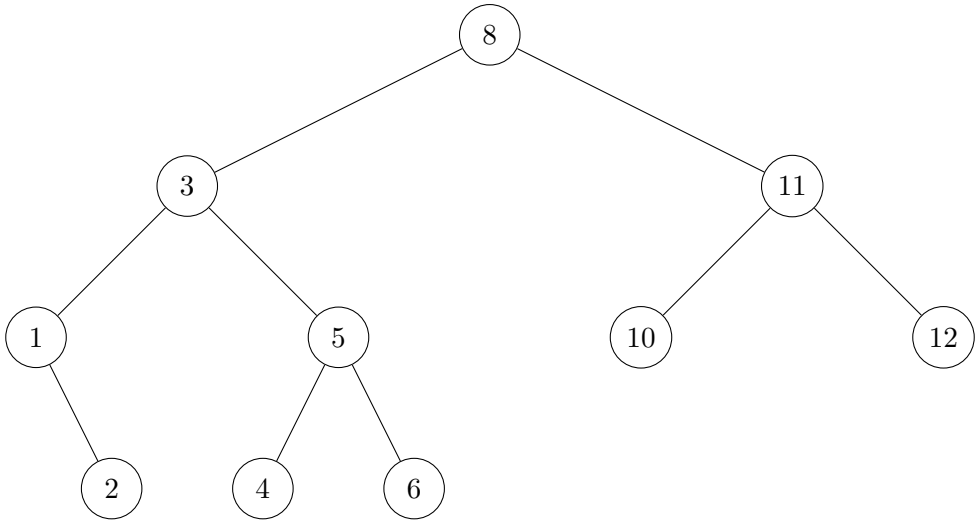


Abbildung 5.1: Suchbaum

Algorithmus 14 Einfügen in einen Suchbaum

```
Prozedur EINFÜGEN(v, D)
    Falls v = NIL dann                          ▷ Leerer Suchbaum wird initialisiert
        Sei v_0 neuer Knoten
        v_0.D := D
        v_0.links := NIL
        v_0.rechts := NIL
        Antworte v_0
    sonst falls v.D.x = D.x dann
        Antworte "Schlüssel existiert bereits"
    sonst falls v.D.x < D.x dann
        v.rechts := EINFÜGEN(v.rechts, D)
        Antworte v
    sonst                                       ▷ v.D.x > D.x
        v.links := EINFÜGEN(v.links, D)
        Antworte v
    Ende Falls
Ende Prozedur
```

Die Vorgehensweise zum Löschen eines Elements wird am besten zunächst an einem Beispiel illustriert. Angenommen wir wollen aus dem in Abbildung 5.1 angegebenen Suchbaum das Element mit dem Schlüssel 3 löschen. Dann wird dadurch zunächst einmal die Baumstruktur zerstört und wir erhalten den in Abbildung 5.2 angegebenen Suchbaum. Eine konservative Methode zur Wiederherstellung eines Suchbaums, d.h. eine die möglichst wenig verändert, besteht darin, ein geeignetes Element an die entstandene Lücke zu verschieben. Dafür geeignete Element sind einerseits das Maximum im linken Teilbaum von 3 sowie andererseits das Minimum im rechten Teilbaum von 3. Wenn wir uns für das Maximum m des linken Teilbaums entscheiden und an die Stelle des gelöschten Elements setzen sind danach alle Schlüssel im neuen linken Teilbaum kleiner als m und alle Schlüssel im rechten Teilbaum sind größer als m da sie ja größer als das gelöschte Element waren und dieses wiederum größer als m. Das Argument für das Minimum

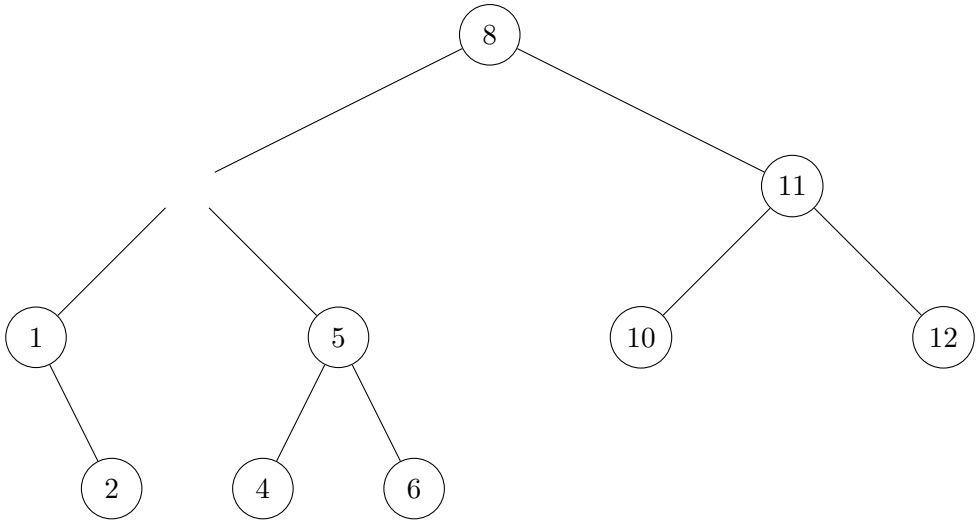


Abbildung 5.2: Suchbaum nach Löschung von 3

des rechten Teilbaums ist analog. In unserem Beispiel erhalten wir also den in Abbildung 5.3 angegebenen Suchbaum. Als Pseudocode ist sieht dieser Algorithmus wie folgt aus:

Algorithmus 15 Löschen aus einem Suchbaum

```
Prozedur LÖSCHEN(v, x)
    Falls v = NIL dann
        Antworte "nicht gefunden"
    sonst falls v.D.x = x dann
        Antworte LÖSCHEWURZEL(v)
    sonst falls v.D.x < x dann
        v.rechts := LÖSCHEN(v.rechts, x)
        Antworte v
    sonst                                          ▷ v.D.x > x
        v.links := LÖSCHEN(v.links, x)
        Antworte v
    Ende Falls
Ende Prozedur
```

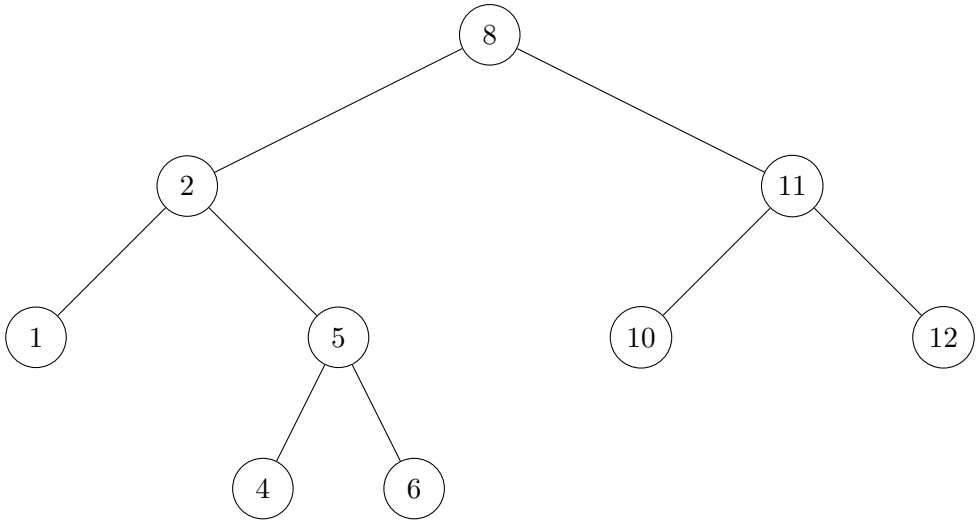


Abbildung 5.3: Suchbaum nach Ersetzung von 3 durch 2

Algorithmus 16 Löschen der Wurzel aus einem Suchbaum

Prozedur LöscheWurzel(v)
  Falls $v.links \neq$ Nil **dann**
    $(D_{\max}, v_l) :=$ ExtrahiereMax($v.links$)
    Sei v' neuer Knoten
    $v'.D := D_{\max}$
    $v'.links := v_l$
    $v'.rechts := v.rechts$
    Antworte v'
  sonst falls $v.rechts \neq$ Nil **dann**
    $(D_{\min}, v_r) :=$ ExtrahiereMin($v.rechts$)
    Sei v' neuer Knoten
    $v'.D := D_{\min}$
    $v'.links := v.links$
    $v'.rechts := v_r$
    Antworte v'
  sonst ▷ v ist ein Blattknoten
    Antworte Nil
  Ende Falls
Ende Prozedur

Algorithmus 17 Finde und lösche Minimum aus Suchbaum

```
Prozedur ExtrahiereMin(v)
    Falls v = Nil dann
        Antworte "Minimum nicht definiert"
    sonst falls v.links = Nil dann                    ▷ v ist Minimum
        Antworte (v.D, v.rechts)
    sonst
        (D_min, v'_l) := ExtrahiereMin(v.links)
        v.links := v'_l
        Antworte (D_min, v)
    Ende Falls
Ende Prozedur
```

Insgesamt erhalten wir also die folgenden Laufzeitkomplexitäten für die Implementierung eines Wörterbuchs als Suchbaum:

1. $M.$*Einfügen*(D) in Zeit $O(h)$

2. $M.$*Löschen*(x) in Zeit $O(h)$

3. $M.$*Suche*(x) in Zeit $O(h)$

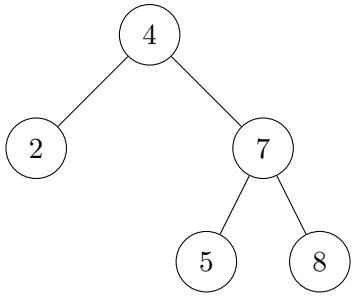
wobei h die Höhe des Baums ist. Das führt zur Frage: Was ist die Höhe des Baums?

Im schlechtesten Fall hat der Baum eine lineare Struktur. Das geschieht wenn die Elemente von M in (aufsteigend oder absteigend) sortierter Reihenfolge eingefügt werden. Dann ist $h = n = |M|$. Im besten Fall ist der Baum ein fast vollständiger Binärbaum, d.h. alle Schichten bis auf die letzte sind vollständig. In diesem Fall ist $h = \Theta(\log n)$.

Wir wollen jetzt auch (einen) Durchschnittsfall analysieren. Auch hier ist a priori nicht klar welche Wahrscheinlichkeitsverteilung betrachtet werden soll. Wir entscheiden uns für die folgende Anwendung: ein Suchbaum wird aus einer Liste $D_1, \ldots, D_n$ mit paarweise verschiedenen Schlüsseln mittels der Einfüge-Operation aufgebaut. Danach verändert er sich nicht mehr und wir suchen nach Elementen aus der Eingabeliste. Wir interessieren uns also für die durchschnittlichen Kosten einer Such-Operation. Nachdem es für diese Anwendung nur auf die Reihenfolge der Schlüssel ankommt, reicht es darauf eine Wahrscheinlichkeitsverteilung zu definieren. Ähnlich wie bei unserer Analyse von Sortieralgorithmen nehmen wir an, dass jede Permutation der Schlüssel die gleiche Wahrscheinlichkeit $\frac{1}{n!}$ hat (Permutationsmodell). Diese Annahme hat übrigens *nicht* zur Folge dass jeder Baum mit n Knoten gleich wahrscheinlich ist, da verschiedene Permutationen den selben Baum erzeugen.

Bevor wir mit der Analyse beginnen ist es nützlich noch einige Begriffe für Bäume einzuführen: Die *Tiefe* $\mathrm{d}(v)$ eines Knoten v in einem Baum T ist die Anzahl der Kanten auf dem eindeutigen Pfad von v zur Wurzel von T. Die Anzahl der Schlüsselvergleiche die zum Auffinden eines Knotens v notwendig sind ist $\mathrm{d}(v)+1$. Für einen Baum $T = (V, E)$ definieren wir die *Pfadlänge* $\mathrm{L}(T)$ von T durch $\mathrm{L}(T) = \sum_{v \in V}(\mathrm{d}(v) + 1)$. Die durchschnittlichen Kosten für die Suche eines in T vorhandenen Schlüssels in T belaufen sich auf $\bar{\mathrm{L}}(T) = \frac{\mathrm{L}(T)}{|T|}$.

Beispiel 5.1. Der Baum



hat zum Beispiel $d(2) = 1$, $d(5) = 2, L(T) = 1 + 2 + 2 + 3 + 3 = 11$ und $\bar{\mathrm{L}}(T) = \frac{11}{5} = 2{,}2$.

Satz 5.1. *Im Permutationsmodell ist der Erwartungswert der Kosten einer Such-Operation* $O(\log n)$.

Beweis. Sei T ein Baum. Falls T leer ist, dann ist $\mathrm{L}(T) = 0$. Falls T nicht leer ist, sei T_l dessen linker Teilbaum und T_r dessen rechter Teilbaum. Dann ist

$$\mathrm{L}(T) = \mathrm{L}(T_\mathrm{l}) + |T_\mathrm{l}| + \mathrm{L}(T_\mathrm{r}) + |T_\mathrm{r}| + 1 = \mathrm{L}(T_\mathrm{l}) + \mathrm{L}(T_\mathrm{r}) + |T|. \qquad (*)$$

Dem Permutationsmodell entsprechend wählen wir uniform verteilt eine Eingabepermutation π von n Datensätzen mit paarweise unterschiedlichen Schlüsseln. Diese Eingabepermutation induziert einen Suchbaum T_π mit Pfadlänge $L(T_\pi)$. Wir schreiben $\mathrm{EL}(n)$ für den Erwartungswert der Pfadlänge von T_π.

Das erste Element der Eingabepermutation ist die Wurzel des Baums. Der linke Teilbaum der Wurzel wird so viele Elemente enthalten wie es D_i in der Eingabepermutation gibt deren Schlüssel kleiner als der der Wurzel ist, der rechte Teilbaum so viele Elemente wie es D_i gibt deren Schlüssel größer ist. Nachdem die Eingabepermutation zufällig war, ist für alle $k \in \{1, \ldots, n\}$ die Wahrscheinlichkeit dass das erste Element den k-t-größten Schlüssel in $D_1, \ldots, D_n$ hat $\frac{1}{n}$. Mit (*) erhalten wir also $\mathrm{EL}(0) = 0$ und, für $n \geq 1$:

$$\begin{aligned}
\mathrm{EL}(n) &= \frac{1}{n}\sum_{k=1}^{n}(\mathrm{EL}(k-1) + \mathrm{EL}(n-k) + n) \\
&= n + \frac{1}{n}\left(\sum_{k=1}^{n}\mathrm{EL}(k-1) + \sum_{k=1}^{n}\mathrm{EL}(n-k)\right) \\
&= n + \frac{2}{n}\sum_{k=0}^{n-1}\mathrm{EL}(k).
\end{aligned}$$

Wir haben also

$$\begin{aligned}
n\mathrm{EL}(n) &= n^2 + 2\sum_{k=0}^{n-1}\mathrm{EL}(k), \\
(n-1)\mathrm{EL}(n-1) &= (n-1)^2 + 2\sum_{k=0}^{n-2}\mathrm{EL}(k), \\
n\mathrm{EL}(n) - (n-1)\mathrm{EL}(n-1) &= 2n - 1 + 2\mathrm{EL}(n-1) \text{ und} \\
\mathrm{EL}(n) &= \frac{2n-1}{n} + \frac{n+1}{n}\mathrm{EL}(n-1).
\end{aligned}$$

Somit ist $\mathrm{EL}(n)$ durch eine lineare Rekursionsgleichung erster Ordnung gegeben. Also

$$\begin{aligned}\mathrm{EL}(n) &=^{\text{Satz 4.1}} \sum_{i=1}^{n} \frac{2i-1}{i} \prod_{j=i+1}^{n} \frac{j+1}{j} = (n+1)\sum_{i=1}^{n} \frac{2i-1}{i(i+1)} \\ &= (n+1)\left(\sum_{i=1}^{n} \frac{3}{i+1} - \sum_{i=1}^{n} \frac{1}{i}\right) = (n+1)\left(\frac{3}{n+1} + \sum_{i=2}^{n} \frac{2}{i} - 1\right) \\ &= 2(n+1)H_n - 3n\end{aligned}$$

wobei $H_n = \sum_{i=1}^{n} \frac{1}{i}$ die n-te harmonische Zahl ist. Die n-te harmonische Zahl kann abgeschätzt werden durch $H_n = \ln n + \gamma + O(\frac{1}{n})$ wobei γ die Euler-Mascheroni Konstante ist und einen Wert von 0,577... hat. Wir erhalten also $\mathrm{EL}(n) = O(n \ln n) = O(n \log n)$ und damit $\frac{\mathrm{EL}(n)}{n} = O(\log n)$. □

Suchbäume verhalten sich also (im Permutationsmodell) im Durchschnittsfall so wie im besten Fall (anders als das beim Einfügesortieren, sh. Kapitel 1, der Fall war). Trotzdem ist diese Lösung noch nicht völlig zufriedenstellend. Wir würden das Wörterbuchproblem gerne mit Operationen lösen die auch im schlechtesten Fall logarithmisch sind, umso mehr als der schlechteste Fall eine sortierte Eingabe ist (was in vielen Anwendungen mit einer Wahrscheinlichkeit von mehr als $\frac{1}{n!}$ auftritt).