Algorithmen und Datenstrukturen (Th. Ottmann und P. Widmayer)

Folien: Suchbäume

Autor: Stefan Edelkamp

Institut für Informatik
Georges-Köhler-Allee
Albert-Ludwigs-Universität Freiburg

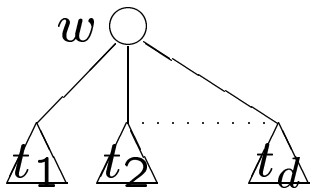
1 Bäume

Idee: Bäume sind verallgemeinerte Listenstrukturen.

Nachfolger: Endlich viele Kinder/Söhne eines direkten Vorgängers/Vaters, i.a. (an)geordnet (erstes, zweites, drittes Kind). Ordnung: max. Anzahl von Kindern.

Darstellung: (ungerichgeter) Verbund von Knoten mit ausgezeichneter Wurzel. Knoten ohne (bzw. mit `null`)-Nachfolger heißen Blätter, sonst innere Knoten.

Rekursive Definition: Bäume mit Ordnung d, Höhe h:
(1) Der aus einem einzigen Knoten bestehende Baum ist ein Baum der Ordnung d. Die Höhe h ist 0.
(2) Sind $t_1, \ldots, t_d$ beliebige Bäume der Ordnung d, so erhält man einen (weiteren) Baum der Ordnung d, indem man die Wurzeln von $t_1, \ldots, t_d$ zu Söhnen einer neugeschaffenen Wurzel w macht. Die Höhe h ist $\max\{h(t_1), \ldots, h(t_d)\} + 1$



Festlegung: $d = 2$ Binärbäume, $d > 2$ Vielwegbäume.

Suchbäume

Annahme: Total geornete Menge von Schlüsseln.

Charakterisierung: Die Schlüssel im linken Teilbaum von p sind sämtlich kleiner als der Schlüssel von p, und dieser ist wiederum kleiner als sämtliche Schlüssel im rechten Teilbaum von p.

Implementation: (Schlüssel ist Ganzzahl)

```
public class Knoten {
  Knoten leftson, rightson;
  int key;  // Hier koennte noch ein Infotype kommen
  Knoten (int key) { this.key = key; }
  public String toString() {
    StringBuffer st = new StringBuffer(""+key);
    return st.toString();
  }
}
```

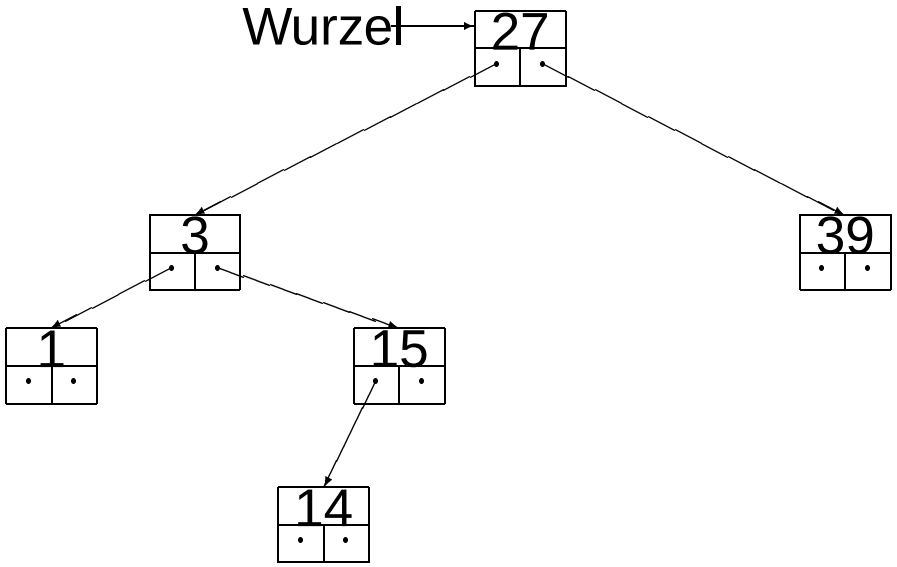
Wörterbuch - Dictionary: Operationen

Suchen/Einfügen/Entfernen - search/insert/delete

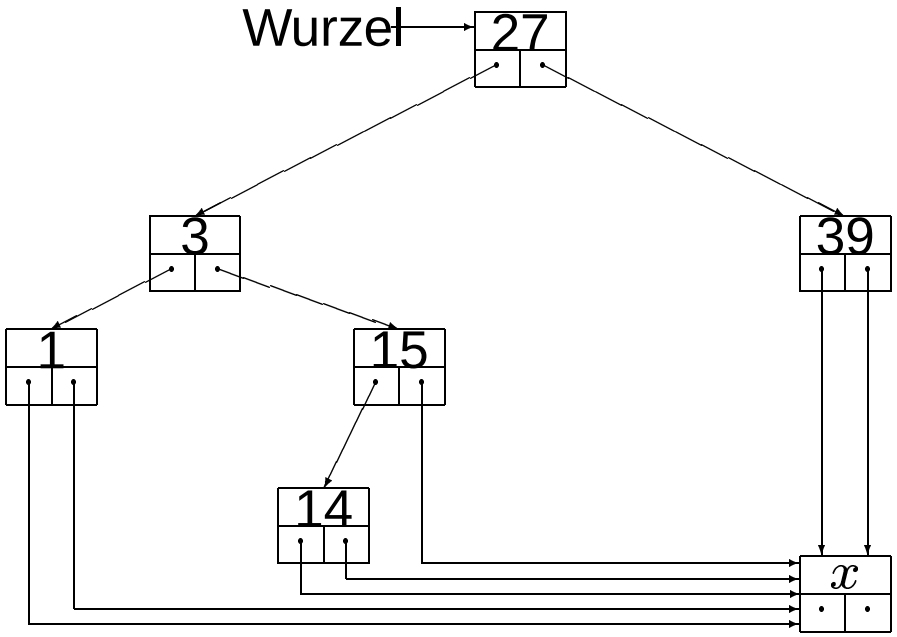
```
public class Baum {
  Knoten Wurzel;
  Baum() { Wurzel = null; }
  public Knoten Suchen(int k) ...
  public void Einfuegen(int k) ...
  public void Entfernen(int k) ...
  public String toString() ...
}
```

Beispiel

Ohne Stopper:



Mit Stopper:



Einfügen/Suchen

Top-Level Aufrufe:

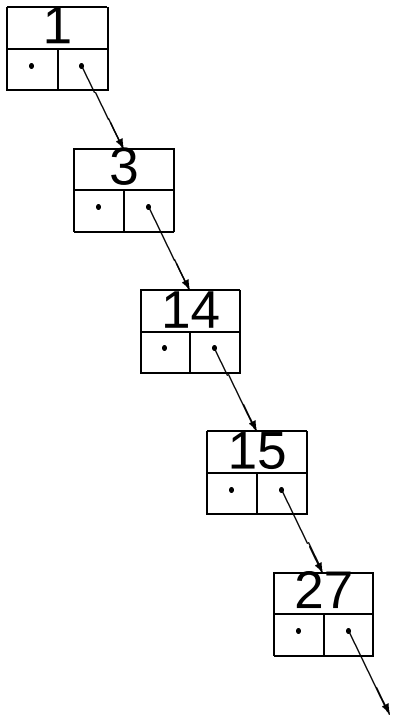
```
public Knoten Suchen(int k) {
  return Suchen(Wurzel,k);
}
public void Einfuegen(int k) {
  Wurzel = Einfuegen(Wurzel,k);
}
```

Rekursive Aufrufe:

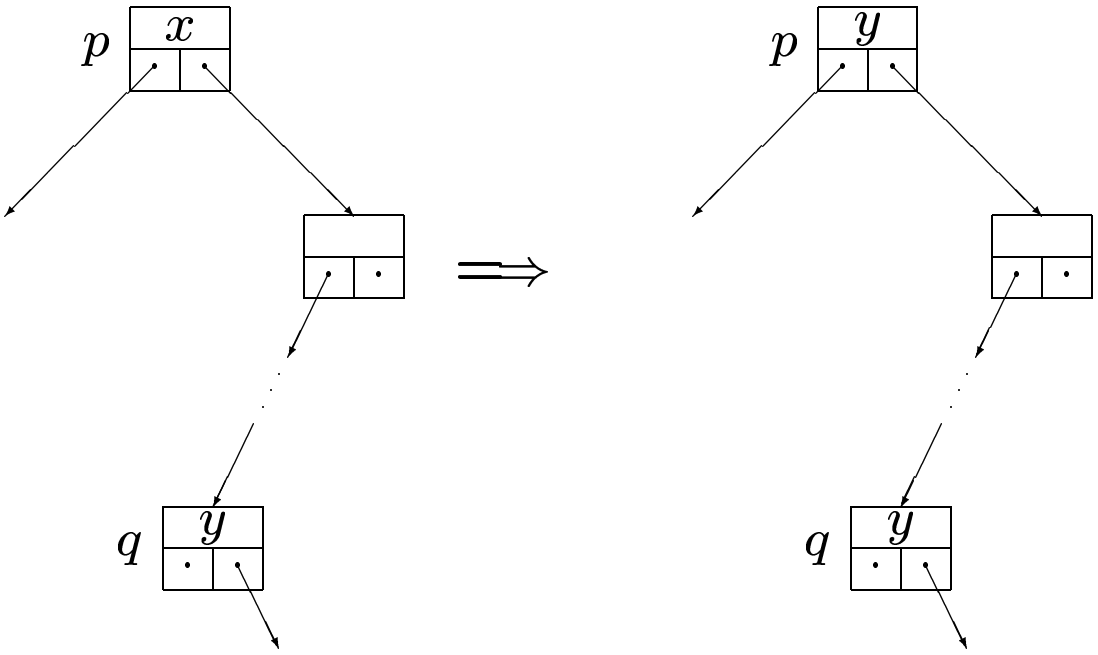
```
Knoten Suchen(Knoten p, int x) {
  if (p == null) return null;
  if (x < p.key)
    return Suchen(p.leftson,x);
  else
    if (x > p.key)
      return Suchen(p.rightson,x);
  return p;
}
Knoten Einfuegen(Knoten p, int k) {
  if (p == null) return new Knoten(k);
  else
    if (k < p.key)
      p.leftson = Einfuegen(p.leftson,k);
    else
      if (k > p.key)
        p.rightson = Einfuegen(p.rightson,k);
      else
        return null; // Schluessel kam vor
  return p;
}
```

Sonderfälle

Entartung:



Löschen eines Knotens:



Löschen

```
public void Entfernen(int k) {
  Wurzel = Entfernen(Wurzel,k);
}
Knoten vatersymnach (Knoten p) {
  if (p.rightson.leftson != null) {
    p = p.rightson;
    while (p.leftson.leftson != null)
      p = p.leftson;
  }
  return p;
}
Knoten Entfernen(Knoten p, int k) {
  if (p == null) return p;
  if (k < p.key)
    p.leftson = Entfernen(p.leftson,k);
  else
    if (k > p.key)
      p.rightson = Entfernen(p.rightson,k);
    else {
      if (p.leftson == null)
        return p.rightson;
      if (p.rightson == null)
        return p.leftson;
      Knoten q = vatersymnach(p);
      if (q == p) {
        p.key = p.rightson.key;
        q.rightson = q.rightson.rightson;
      }
      else {
        p.key = p.leftson.key;
        q.leftson = q.leftson.rightson;
      }
    }
  return p;
}
```

2 Durchlaufreihenfolgen

Voraussetzung:

Sei $T(r)$ ein geordneter Baum (die Kinder eines Knotens sind linear geordnet) mit Wurzel r und Kindern $v_1, \ldots, v_k$ (dabei ist $k = 0$ möglich).

Definition:

Dann ist der Postorderdurchlauf $post(T)$ definiert als $post(T(v_1)) \ldots post(T(v_k))\ r$. Entsprechend sind der Inorderdurchlauf $in(T)$ als $in(T(v_1))\ r\ in(T(v_2)) \ldots in(T(v_k))$, der Preorderdurchlauf $pre(T)$ als $r\ pre(T(v_1)) \ldots pre(T(v_k))$ und der Levelorderdurchlauf als $lev(T)$ als $r\ lev(T(v_1))\ r\ lev(T(v_2))\ \ldots\ r\ lev(T(v_k))\ r$ festgelegt.

Eigenschaften

Beispiel:

Preorder: D,E,B,H,K,L,I,F,G,C,A
Postorder: A,B,D,E,C,F,H,I,K,L,G
Inorder: D,B,E,A,H,F,K,I,L,C,G
Levelorder: A,B,D,B,E,B,A,C,F,H,F,I,K,I,L,I,C,G,C,A

1. Pre- In-, und Postorder haben jeweils die Länge n.
2. Pre- In-, und Postorder allein nicht eindeutig.
3. Levelorder hat die Länge $2n - 1$
4. Levelorder ist eindeutig. (Induktion: $|lev(T)| = 1$ ist klar. Ansonsten ist der erste Knoten Wurzel. Zwischen den verschiedenen Darstellungen der Wurzel finden wir die Levelorder der Teilbäume.)
5. Preorder und Inorder beschreiben einen Baum T nicht eindeutig. Gegenbeispiel:
   $pre(T_1) = pre(T_2) = 1, 2, 3, 4, 5, 6$
   $in(T_1) = in(T_2) = 2, 1, 4, 3, 5, 6$
   Einen binären Baum übrigens schon (Übung)
6. Preorder und Postorder beschreiben T eindeutig (Übung).

Implementation

```
String Inorder(Knoten p) { // Symm. Reihenfolge
  if (p == null) return new String("");
  StringBuffer st = new
    StringBuffer(Inorder(p.leftson) +
                 "(" + p.toString() + ")" +
                 Inorder(p.rightson));
  return st.toString();
}
String Preorder(Knoten p) { // Hauptreihenfolge
  if (p == null) return new String("");
  StringBuffer st = new
    StringBuffer("(" + p.toString() + ")" +
                 Preorder(p.leftson) +
                 Preorder(p.rightson));
  return st.toString();
}
String Postorder(Knoten p) { // Nebenreihenfolge
  if (p == null) return new String("");
  StringBuffer st = new
    StringBuffer(Postorder(p.leftson) +
                 Postorder(p.rightson) +
                 "(" + p.toString() + ")");
  return st.toString();
}
public String toString() {
  StringBuffer st = new StringBuffer(
    "Preorder : " + Preorder(Wurzel) + "\n" +
    "Inorder  : " + Inorder(Wurzel) + "\n" +
    "Postorder: " + Postorder(Wurzel));
  return st.toString();
}
```

Sortieren mit natürlichen Suchbäumen

Idee: Baue für die Eingabefolge einen natürlichen Suchbaum auf und gebe die Schlüssel in symmetrischer Reihenfolge (Inorder) aus.

Bemerkung: Abhängig von der Eingabereihenfolge kann der Suchbaum degenerieren.

Komplexität: Abhängig von der internen Pfadlänge

Schlechtester Fall: Vorsortierung $\Rightarrow \Omega(n^2)$ Schritte.

Bester Fall: Ausgeglichener Suchbaum (Analyse ähnlich wie die des mittleren Falles von binärer Suche) $\Rightarrow$ $O(n \log n)$ — mit Faktor 1 vor $n \log n$

Mittlerer Fall: Erwartungswert für interne Pfadlänge ist $\approx 1.386 n \log_2 n - 0.846n + O(\log n)$.

Interne Pfadlänge

(0): Ist t ein Blatt, so ist $I(t) = 0$.
(1): Ist t ein Baum mit linkem Teilbaum mit Wurzel t_l und rechtem Teilbaum mit Wurzel t_r, so ist

$I(t) := I(t_l) + I(t_r) +$ Zahl der inneren Knoten von t.

$$I(t) = \sum_{\substack{p \\ p \text{ innerer} \\ \text{Knoten von } t}} (\text{Tiefe}(p) + 1)$$

$EI(N)$: Erwartungswert für die interne Pfadlänge eines zufällig erzeugten binären Suchbaumes mit N inneren Knoten, so gilt

$$\begin{aligned} EI(0) &= 0, \quad EI(1) = 1, \\ EI(N) &= \frac{1}{N} \sum_{k=1}^{N} (EI(k-1) + EI(N-k) + N) \\ &= N + \frac{1}{N}\left(\sum_{k=1}^{N} EI(k-1) + \sum_{k=1}^{N} EI(N-k)\right) \end{aligned}$$

Satz:
$EI(N) \approx 1.386 N \log_2 N - 0.846N + O(\log N)$.

Beweis:

I) QUICKSORT Average-Case ($+2N$)

II)

$$EI(N+1) \;=\; (N+1) + \frac{2}{N+1} \cdot \sum_{k=0}^{N} EI(k),$$

und daher

$$(N+1) \cdot EI(N+1) \;=\; (N+1)^2 + 2 \cdot \sum_{k=0}^{N} EI(k)$$

$$N \cdot EI(N) \;=\; N^2 + 2 \cdot \sum_{k=0}^{N-1} EI(k).$$

Aus den beiden letzten Gleichungen folgt

$$\begin{aligned} (N+1)EI(N+1) - N \cdot EI(N) &= 2N + 1 + 2 \cdot EI(N) \\ (N+1)EI(N+1) &= (N+2)EI(N) + 2N + 1 \\ EI(N+1) &= \frac{2N+1}{N+1} + \frac{N+2}{N+1} EI(N). \end{aligned}$$

Nun zeigt man leicht durch vollständige Induktion über N, daß für alle $N \varepsilon 1$ gilt:

$$EI(N) = 2(N+1)H_N - 3N$$

Dabei bezeichnet $H_N = 1 + \frac{1}{2} + \ldots + \frac{1}{N}$ die N-te harmonische Zahl, die wie folgt abgeschätzt werden kann:

$$H_N = \ln N + \gamma + \frac{1}{2N} + O(\frac{1}{N^2})$$

Dabei ist $\gamma = 0.5772\ldots$ die sogenannte Eulersche Konstante.

Damit ist

$$EI(N) = 2N \ln N - (3-2\gamma)\cdot N + 2 \ln N + 1 + 2\gamma + O(\frac{1}{N})$$

und daher

$$\begin{aligned}
\frac{EI(N)}{N} &= 2 \ln N - (3-2\gamma) + \frac{2 \ln N}{N} + \ldots \\
&= \frac{2}{\log_2 e} \cdot \log_2 N - (3-2\gamma) + \frac{2\, ln N}{N} + \ldots \\
&= \frac{2 \log_{10} 2}{\log_{10} e} \cdot \log_2 N - (3-2\gamma) + \frac{2 \ln N}{N} + \ldots \\
&\approx 1.386 \log_2 N - (3-2\gamma) + \frac{2 \ln N}{N} + \ldots
\end{aligned}$$