Wörterbücher

- Definition Wörterbuch: dynamische Menge an Elementen, die über Schlüssel definiert sind:
  Paare (Schlüssel, Wert)
- Operationen
  - Element *einfügen*
  - Element *löschen*
  - Nach Element mit gegebenem Schlüssel *suchen*
  - Alle Elemente nach Schlüssel sortiert *aufzählen*

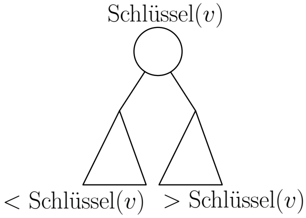
Binäre Suchbäume

- Mögliche Implementierung: Binärer Suchbaum
  (Englisch: Binary Search Tree, kurz BST)
- Zeiger und Felder von Knoten x
  - *Parent(x)*: Vaterknoten von x
  - *Left(x)*: linkes Kind von x
  - *Right(x)*: rechtes Kind von x
  - *Schlüssel(x)*: Schlüssel von x
  - Werte
- Alle nicht vorhandenen Zeiger werden auf NULL gesetzt

Binäre Suchbäume

Ein binärer Suchbaum ist ein binärer Baum mit folgender Eigenschaft:

- *Schlüssel(v)*$<$*Schlüssel(w)* $\forall$*w* im rechten Teilbaum von *v*
- *Schlüssel(v)*$>$*Schlüssel(w)* $\forall$*w* im linken Teilbaum von *v*



Aufzählen

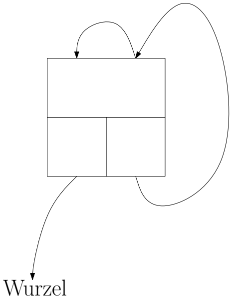
Algorithm 1: Aufzählen(Knoten *x*)

```
if x ≠ NULL then
    Aufzählen(Left(x))
    print x
    Aufzählen(Right(x))
end
```

Sentinel

Wächter oder Sentinel

- Spezieller Knoten, der kein Element permanent abspeichert
- Alle NULL Zeiger zeigen jetzt zum Sentinel
- Sentinel ist dereferenzierbar und erlaubt, Spezialfälle zu vermeiden
- Wir nennen diesen Knoten im Folgenden TNULL



Suche

Algorithm 2: Suche(Knoten *x*, Schlüssel *k*)

```
Schlüssel(TNULL) = k
Sentinel-Suche(Knoten x, Schlüssel k)
```

Algorithm 3: Sentinel-Suche(Knoten *x*, Schlüssel *k*)

```
if (k = Schlüssel(x)) then
    if (x != TNULL) then
        return x
    end
    else
        return NULL
    end
end
if (k < Schlüssel(x)) then
    return Sentinel-Suche(Left(x), k)
end
else
    return Sentinel-Suche(Right(x), k)
end
```

Grundidee Einfügen

Algorithm 4: Einfügen(Knoten x von Baum T, Knoten z)

```
if (x = TNULL) then
    /*Füge z als Blatt in den Baum an der Stelle von x ein (speichere
    im Sentinel / Member-Variablen der Klasse BST Informationen
    zum Aktualisieren von Zeigern)*/
    return
end
else if (Schlüssel(z) < Schlüssel(x)) then
    Einfügen(Left(x), z)
end
else if (Schlüssel(z) > Schlüssel(x)) then
    Einfügen(Right(x), z)
end
else
    /*Schlüssel schon in T*/
    return
end
```

Grundidee Löschen

Drei Fälle beim Löschen von x

- x ist Blattknoten - einfach entfernen
- x hat ein Kind -

  x entfernen und Kind von x mit Vater von x verbinden
- x hat zwei Kinder -
  - Nachfolger y von x (Knoten mit nächst kleinstem Schlüssel nach x) hat nur ein Kind
  - Ersetze x durch y und lösche y

Laufzeiten

n = Anzahl der Knoten im Baum, h = Höhe des Baumes

- Aufzählen: $O(n)$
- Suche: Algorithmus durchläuft maximal einen Pfad von der Wurzel zu einem (leeren) Blatt: $O(h)$
- Einfügen: analog zur Suche $O(h)$
- Löschen: analog zur Suche (Suche des Nachfolgers) $O(h)$

Laufzeiten

- Die besten Laufzeiten werden erreicht, wenn der Baum *balanciert* ist (d.h. alle Teilbäume derselben Tiefe haben gleich viele Knoten).
- Wie können wir einen balancierten binären Suchbaum konstruieren?