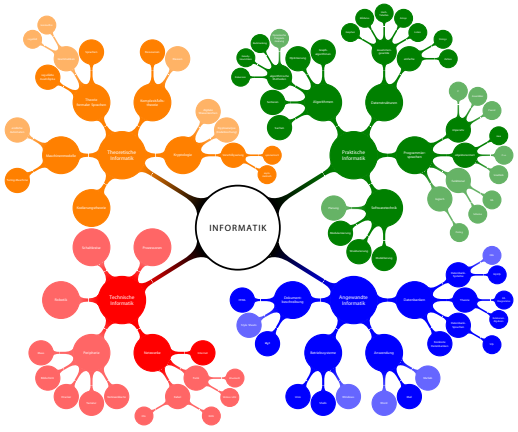
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 28
Suchbäume

Suchet und ihr werdet finden

Vorlesung Einführung in die Informatik 1 vom 11. Februar 2014 von Till Tantau



Lernziele von Kapitel 28

1. Konzept des Suchbaums verstehen
2. Basisoperationen auf Suchbäumen verstehen
3. Vor- und Nachteile von Arrays, Listen und Suchbäumen beurteilen können

Gliederung von Kapitel 28

Die Mitgliederliste der Fachschaft.

Problemstellung

- Eine Fachschaft möchte ihre Mitglieder verwalten.
- Zu jedem Mitglied werden Daten gespeichert; beispielsweise soll für jedes Mitglied die E-Mail-Adresse gespeichert werden.
- Man möchte nun (schnell) nach der E-Mail-Adresse einer Person suchen können, sowie neue Personen anlegen und Personen löschen können.

Die Klasse für die Mitglieder lautet:

```
class Studie {
  String name;
  String email;
}
```

Was unsere Datenstruktur können soll.

Wir suchen eine möglichst gute Implementationen einer Klasse mit folgenden Methoden:

```
class Map {

  Studie search (String name);
  // Sucht nach einem Studenten-Objekt
  // anhand eines Namens. Wenn es ein solches nicht
  // gibt, soll null zurückgegeben werden.

  void add (Studie s);
  // Fügt einen Studenten in die Datenstruktur ein.

  void remove (String name);
  // Löscht einen Studenten anhand seines Namens.
}
```

Arrays und Listen als Implementationen haben Nachteile

- Wir können einen *sortierten Array* benutzen:
  - Suchen dauert $O(\log n)$.
  - *Löschen und Einfügen dauern $O(n)$.*
- Wir können eine *unsortierte Liste* benutzen:
  - Löschen und Einfügen dauern $O(1)$.
  - *Suchen dauert $O(n)$.*

Keine der Implementationen scheint besonders geeignet. Da aber das Suchen viel öfter vorkommt als Einfügen und Löschen, würden wir wohl einen Array benutzen.

Auf dem Weg zum Suchbaum.

Was macht Suchen in sortierten Arrays schnell?

Suchen ist schnell, da wir bei der binären Suche in jedem Schritt den Suchraum halbieren können.

Was macht Einfügen und Löschen langsam?

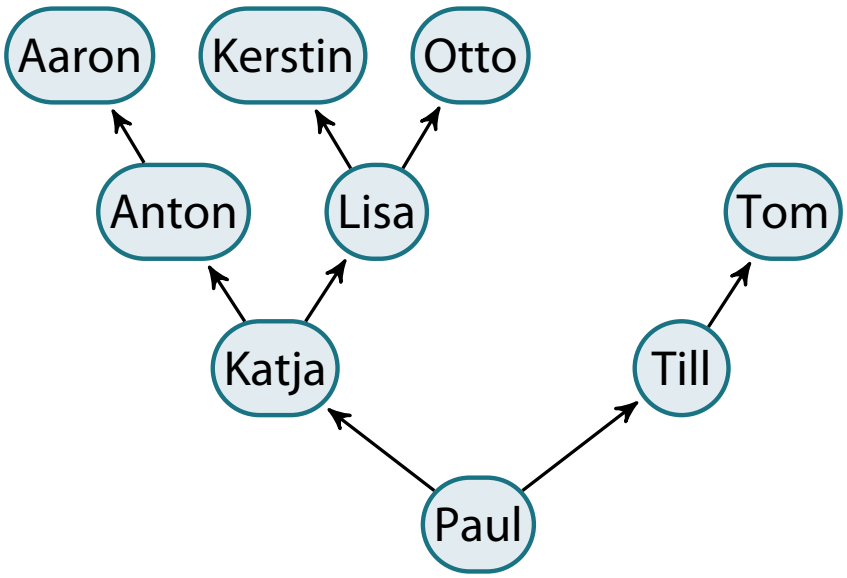
Arrays sind zu starr. Erst durch Verweise und Zellen wie bei Listen wird man flexibler.

Bei einem *Suchbaum* versucht man, die Vorteile von sortierten Listen und von Arrays zu vereinen.

Die Idee des Suchbaumes

- Bei einem Suchbaum nimmt man (wie bei einem sortierten Array) an, dass sich je zwei Studenten *vergleichen* lassen.
- Die Studenten werden aber so eingefügt, dass für jeden Knoten Folgendes gilt:
  1. Alle Studenten im *linken Unterbaum* sind *kleiner* als der Student des Knotens.
  2. Alle Studenten im *rechten Unterbaum* sind *größer oder gleich* dem Studenten des Knotens.

Beispiel eines Suchbaumes.



Suchbäume implementiert man genau wie normale Bäume.

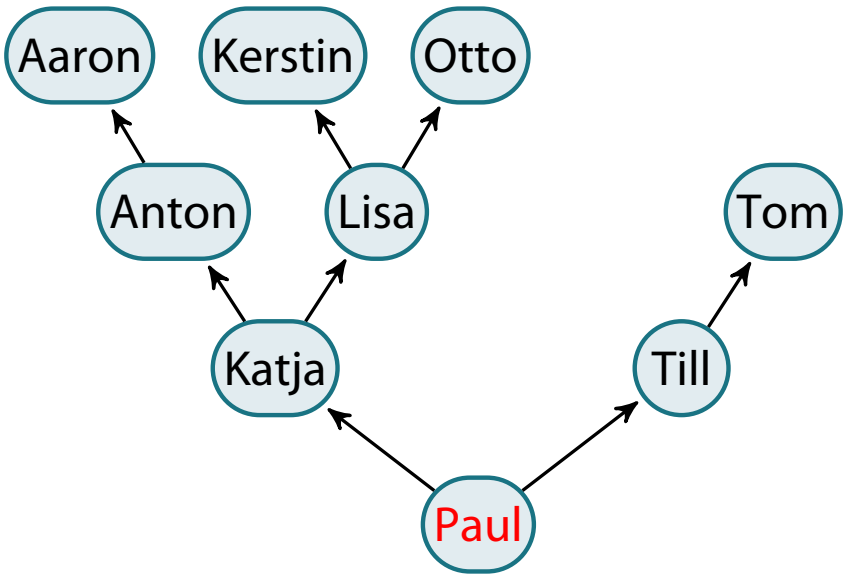

```
class Node {
  // Verweis auf den zum Knoten gehörende Studenten
  Studie studie;

  // Verweise auf die Wurzeln der Kinder
  Node smallerChildren;
  Node largerChildren;

  // Einfacher Konstruktor (Kinder sind automatisch null):
  Node (Studie s) { this.studie = s; }
}

class StudieTree {
  // Die Wurzel ist das einzige Attribut
  Node root;

  // Die drei Operationen
  Studie search (String name) {...}
  void   add    (Studie s)    {...}
  void   remove (String name) {...}
}
```

Wie sucht man in Suchbäumen?

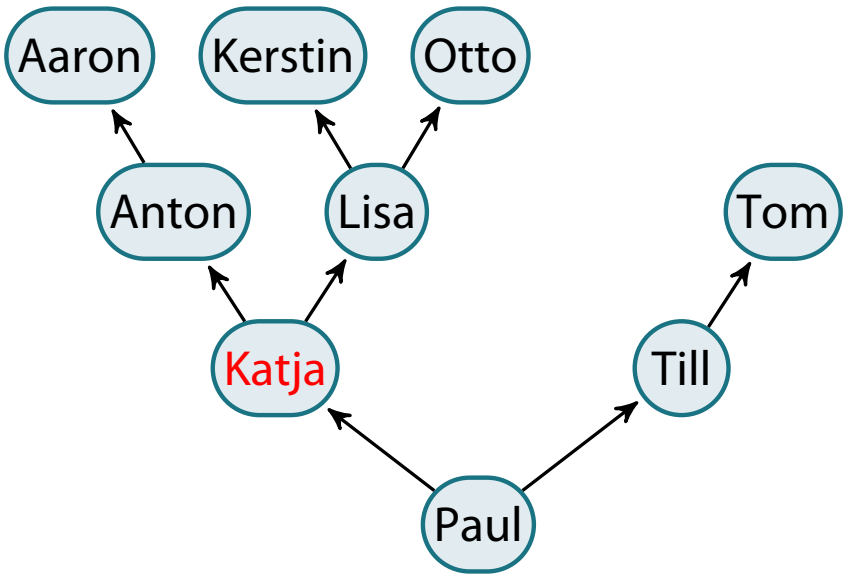
Wir gehen ähnlich einer binären Suche vor:

1. Wir vergleichen den zu suchenden Namen mit dem Namen am aktuellen Knoten.
2. Sind sie *gleich*, so sind wir fertig.
3. Ist der zu suchende Namen *kleiner*, so suchen wir im *linken* Teilbaum weiter.
4. Ist der zu suchende Namen *größer*, so suchen wir im *rechten* Teilbaum weiter.

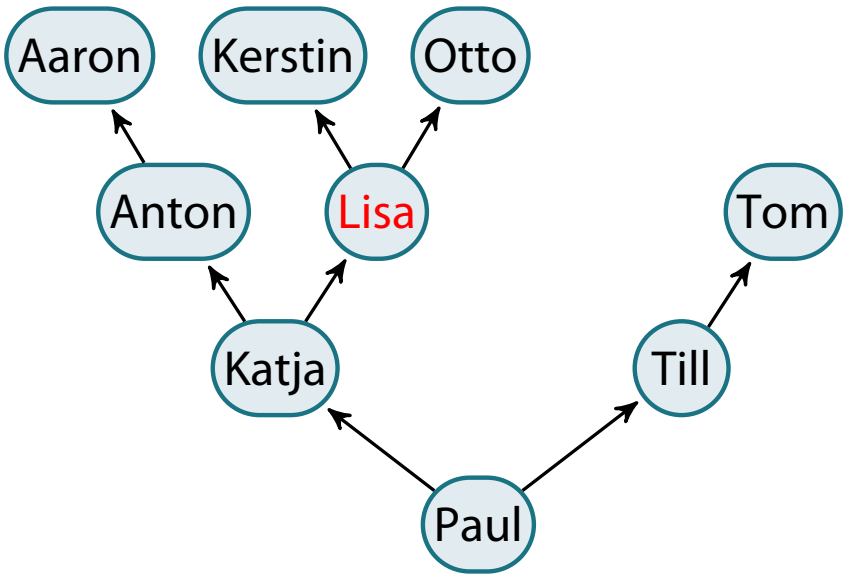
Suche nach *Otto*



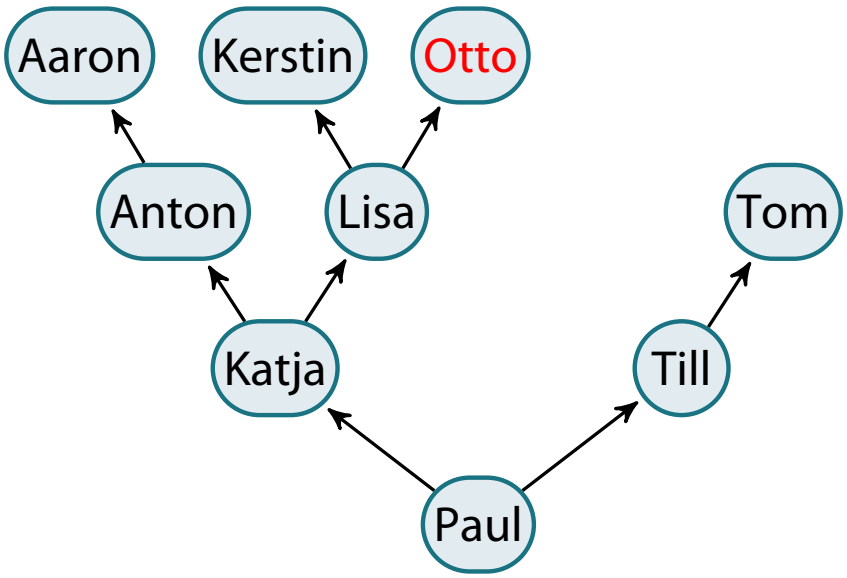
Suche nach *Otto*



Suche nach *Otto*



Suche nach *Otto*



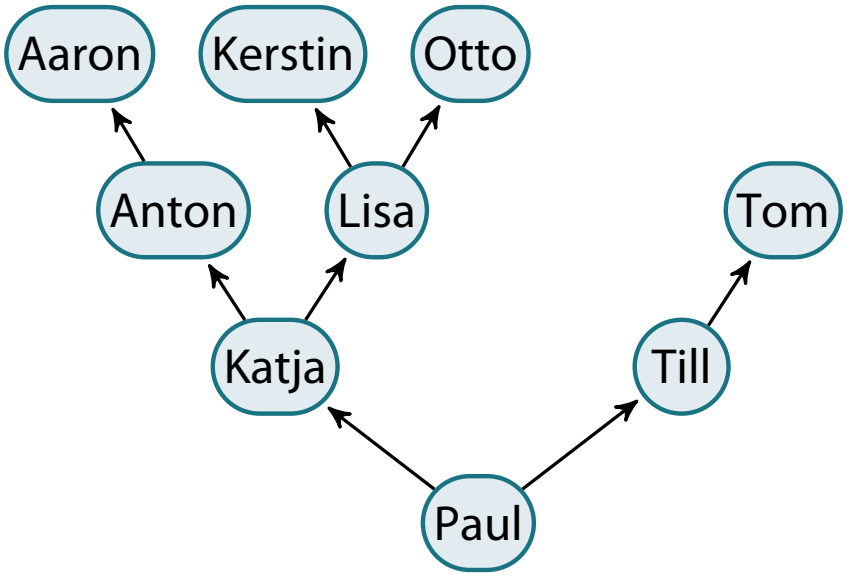
Java-Code der Such-Methode


```
class StudieTree {
  Studie search (String find_me) {
    // Aufruf der eigentlichen rekrusiven Suche:
    return recursiveSearch (find_me, this.root);
  }

  Studie recursiveSearch (String find_me, Node node) {
    if (node == null) { // Baum ist leer, nichts gefunden
      return null;
    }
    else if (node.studie.name.equals(find_me)) {
      // Bingo!
      return node.studie;
    }
    else if (node.studie.name.compare(find_me) < 0) {
      // Mache mit den kleineren Kindern weiter
      return
        recursiveSearch (find_me, node.smallerChildren);
    }
    else { // Mache mit den größeren Kindern weiter
      return
        recursiveSearch (find_me, node.largerChildren);
    }
  }
}
```

Einfügen von Knoten

Zur Diskussion

Wohin sollten wir *Peter* und wohin sollten wir dann *Petra* einfügen?



Wie fügt man in einen Suchbaum ein?

- Man fügt neue Knoten immer als Blätter ein (und nicht irgendwie in der Mitte).
- Dabei gibt es immer genau einen Ort, wo man den Knoten als Blatt einfügen kann,
- den man wie beim Such-Algorithmus findet:
  - Ist der einzufügende Name *kleiner* als der des aktuellen Knoten, mache *links* weiter.
  - Ist der einzufügende Name *größer* als der des aktuellen Knoten, mache *rechts* weiter.

Wir bauen einen Suchbaum mit den Namen der Studenten in den ersten zwei Reihen auf.

Java-Code zum Einfügen in einen Suchbaum.

Add-Methode der Verwaltungsklasse.


```
class StudieTree {
  ...
  void add (Studie s)
  {
    if (this.root == null) {
      this.root = new Node (s);
    }
    else {
      recursiveAdd (s, this.root);
    }
  }
```

Java-Code zum Einfügen in einen Suchbaum.

Add-Methode der Verwaltungsklasse.


```
  void recursiveAdd (Studie s, Node node)
  {
    if (node.studie.name.compareTo(s.name) < 0)
    {
      // s muss nach links
      if (node.smallerChildren == null) {
        node.smallerChildren = new Node(s);
      }
      else {
        recursiveAdd(s, node.smallerChildren);
      }
    }
    else
    {
      // s muss nach rechts
      if (node.largerChildren == null) {
        node.largerChildren = new Node(s);
      }
      else {
        recursiveAdd(s, node.largerChildren);
      }
    }
  }
}
```

Löschen von Knoten

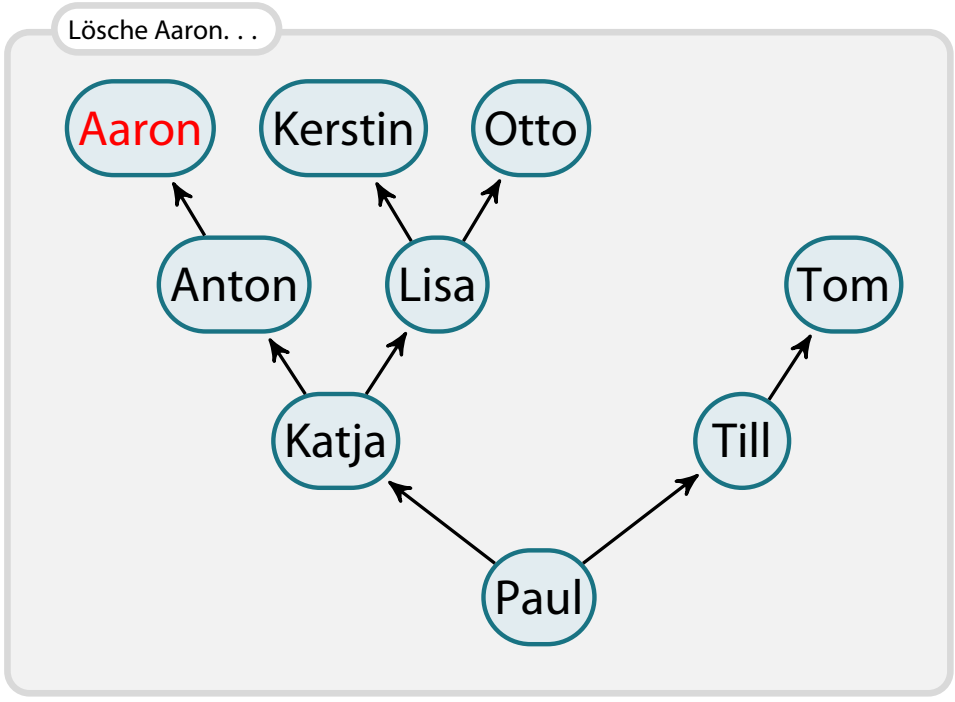
Zur Diskussion

- Es ist leicht, ein Element zu löschen, das ein Blatt ist.
- Viel schwieriger ist es aber, einen inneren Knoten zu löschen.
- Was kann man tun?

Löschen eines Knotens

Fall 1: Blätter

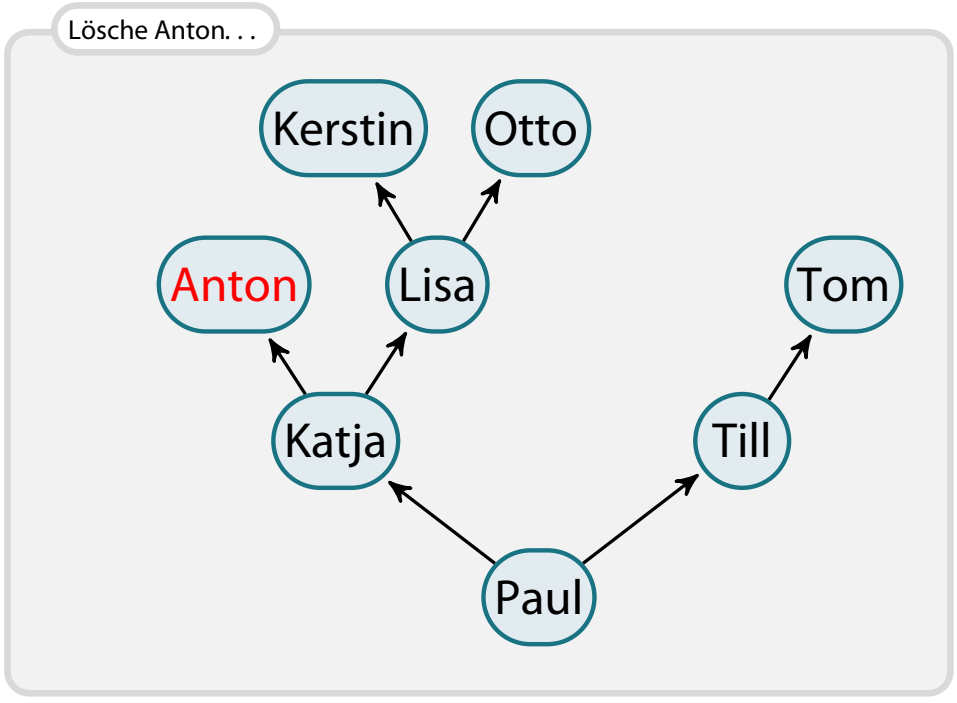
Ein *Knoten ohne Kinder* (ein Blatt) kann einfach gelöscht werden.



Löschen eines Knotens

Fall 1: Blätter

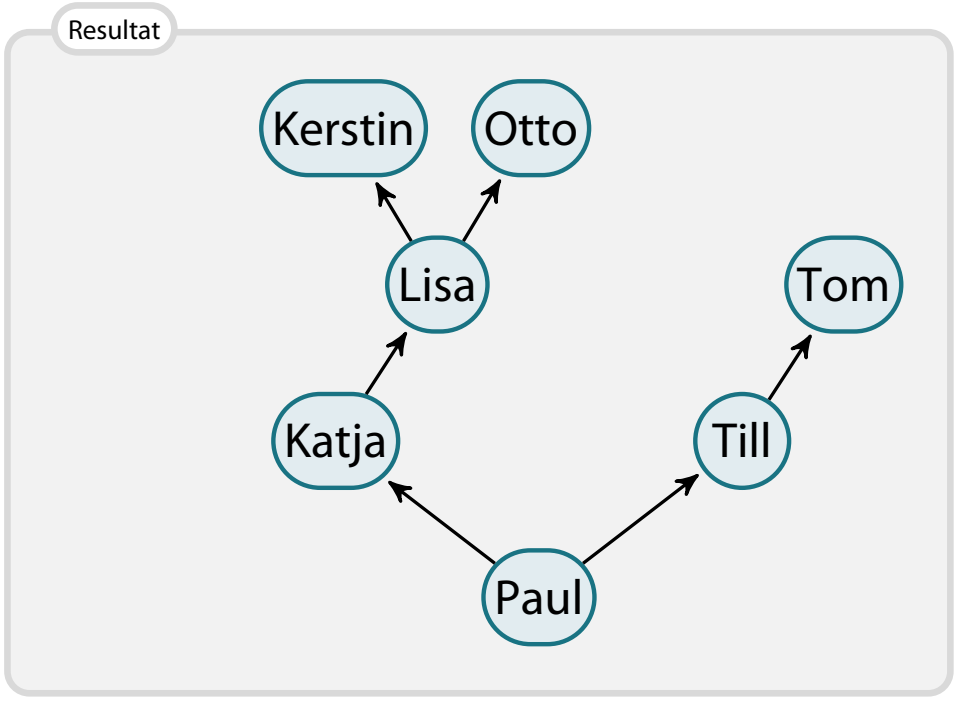
Ein *Knoten ohne Kinder* (ein Blatt) kann einfach gelöscht werden.



Löschen eines Knotens

Fall 1: Blätter

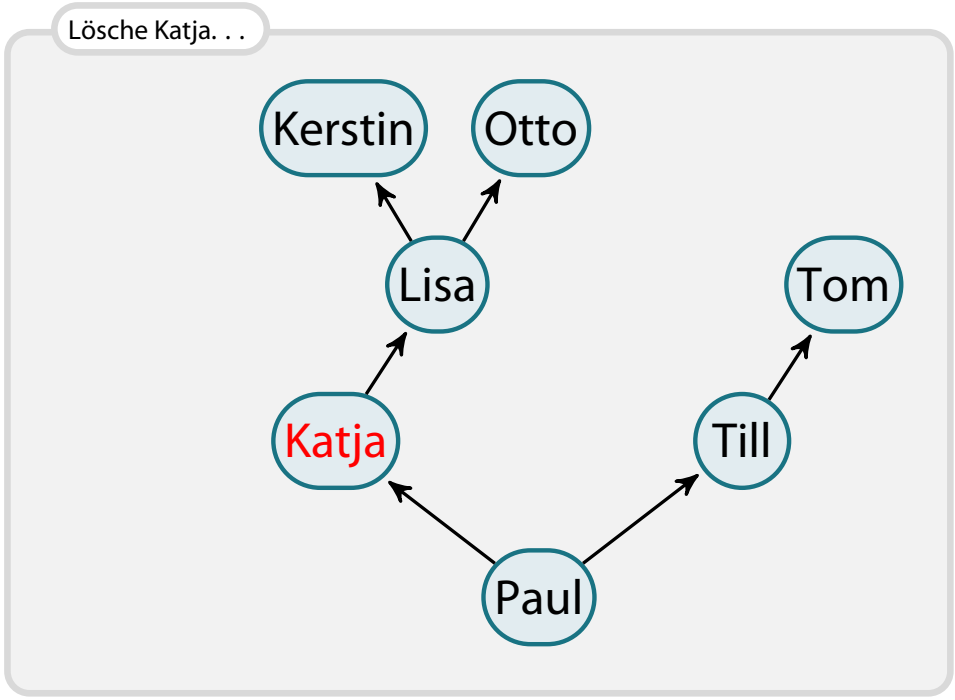
Ein *Knoten ohne Kinder* (ein Blatt) kann einfach gelöscht werden.



Löschen eines Knotens

Fall 2: Knoten mit nur einem Kind

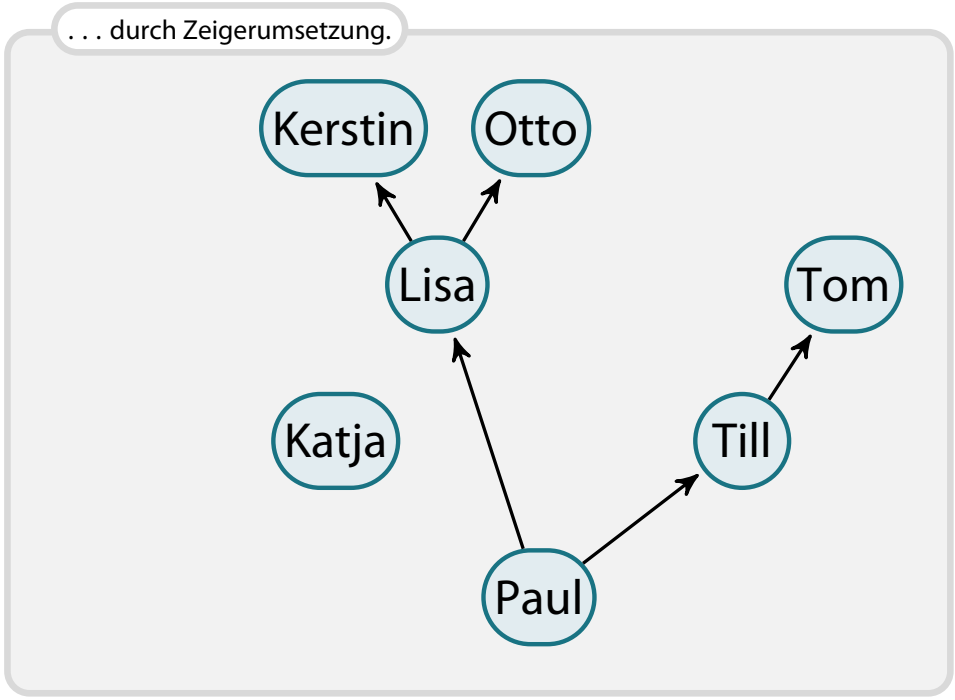
Bei einem *Knoten mit einem Kind* nimmt das Kind den Platz des Knotens ein.



Löschen eines Knotens

Fall 2: Knoten mit nur einem Kind

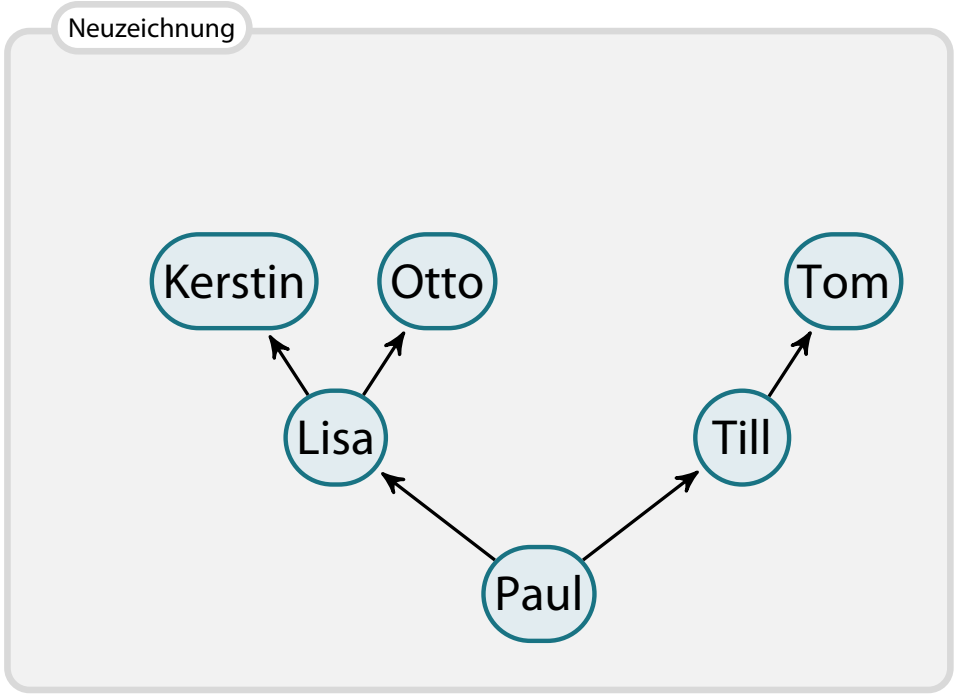
Bei einem *Knoten mit einem Kind* nimmt das Kind den Platz des Knotens ein.



Löschen eines Knotens

Fall 2: Knoten mit nur einem Kind

Bei einem *Knoten mit einem Kind* nimmt das Kind den Platz des Knotens ein.

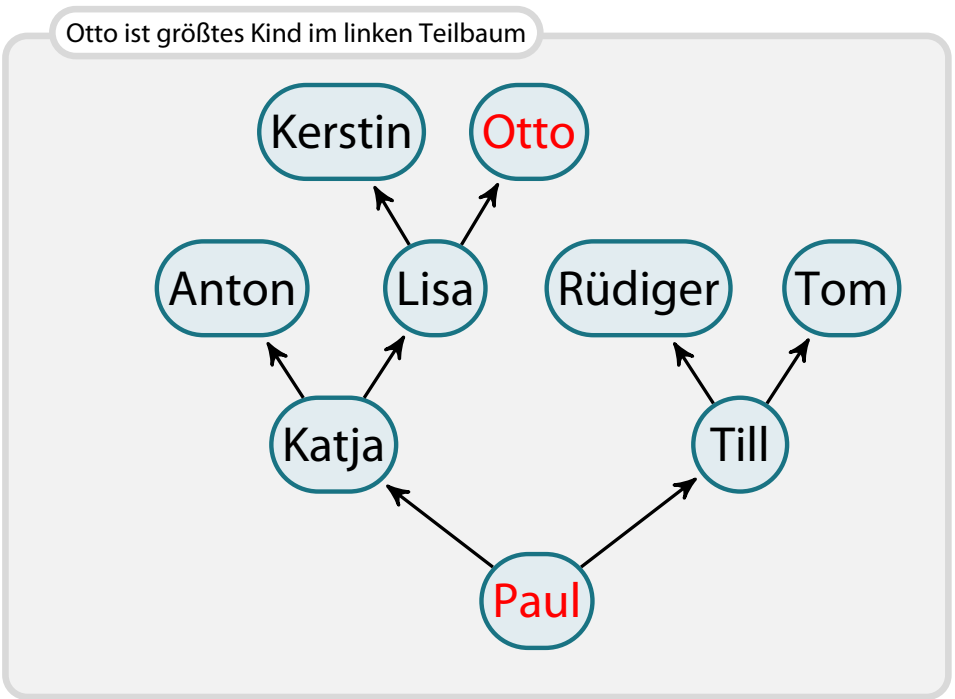


Löschen von *Paul*

Fall 3: Knoten mit zwei Kindern

Löschen eines *Knoten mit zwei Kindern*:

1. Finde den größten Knoten im linken Teilbaum.
2. Merke den Inhalt.
3. Lösche den gefunden Knoten.
4. Ersetze Inhalt des zu löschenden Knotens durch gemerkten.

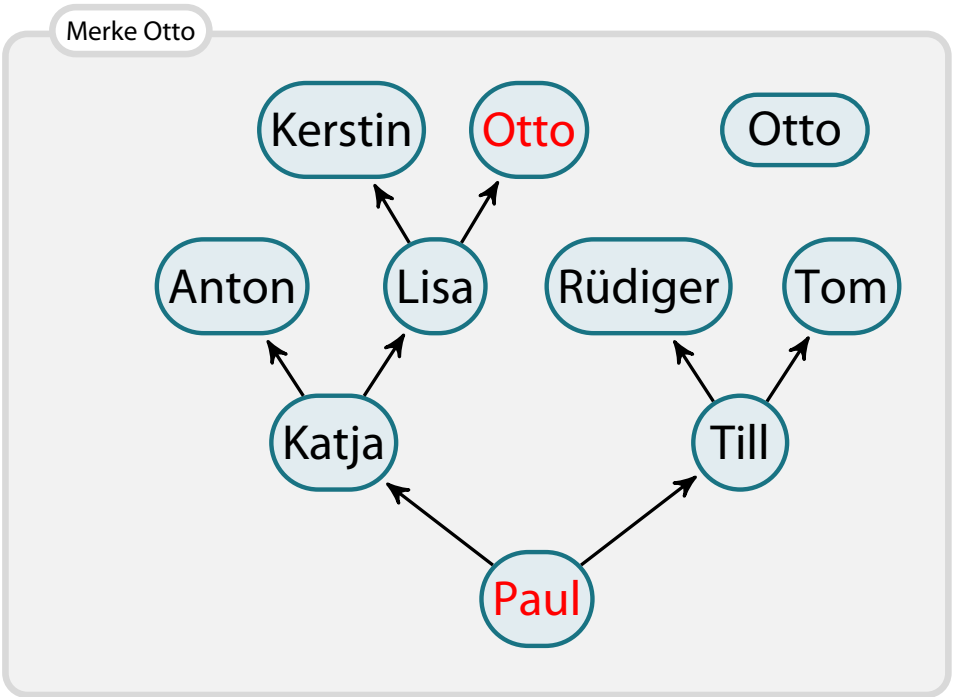


Löschen von *Paul*

Fall 3: Knoten mit zwei Kindern

Löschen eines *Knoten mit zwei Kindern*:

1. *Finde den größten Knoten im linken Teilbaum.*
2. *Merke den Inhalt.*
3. Lösche den gefunden Knoten.
4. Ersetze Inhalt des zu löschenden Knotens durch gemerkten.

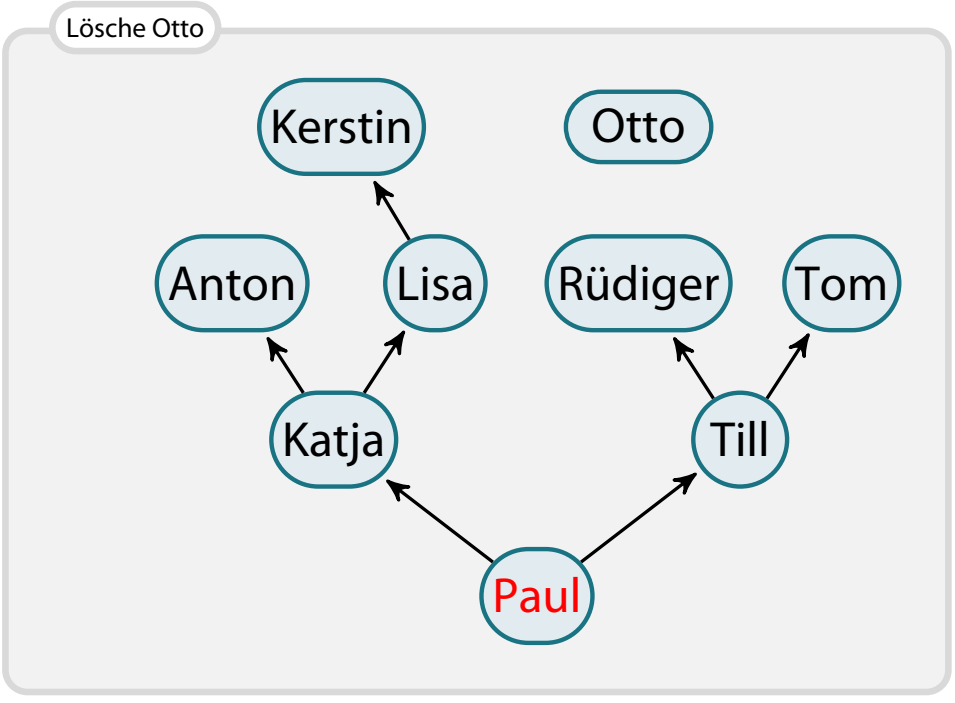


Löschen von *Paul*

Fall 3: Knoten mit zwei Kindern

Löschen eines *Knoten mit zwei Kindern*:

1. Finde den größten Knoten im linken Teilbaum.
2. Merke den Inhalt.
3. *Lösche den gefunden Knoten.*
4. Ersetze Inhalt des zu löschenden Knotens durch gemerkten.

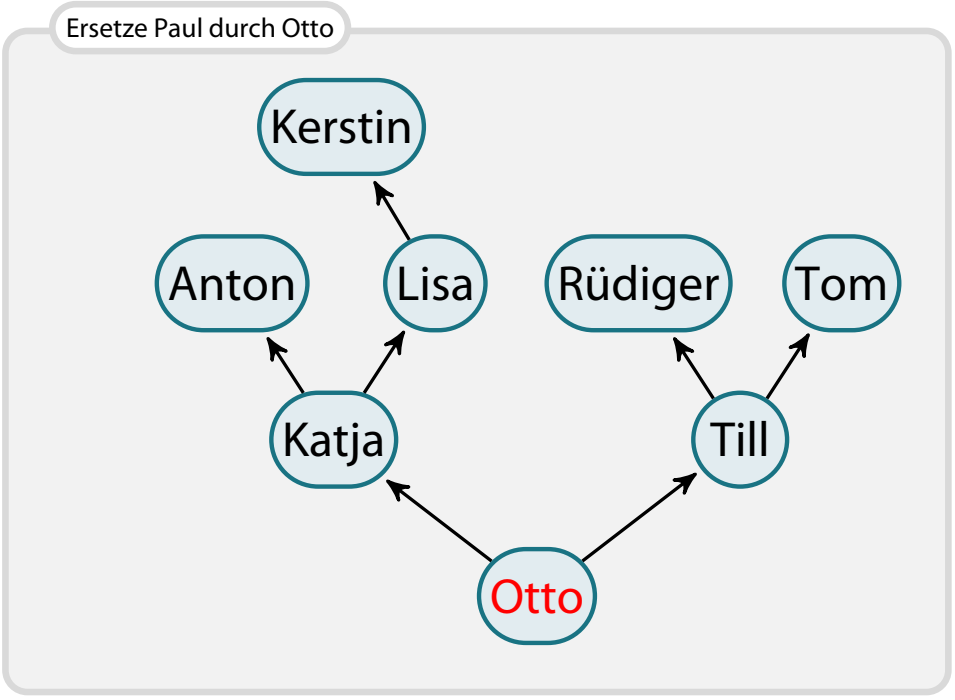


Löschen von *Paul*

Fall 3: Knoten mit zwei Kindern

Löschen eines *Knoten mit zwei Kindern*:

1. Finde den größten Knoten im linken Teilbaum.
2. Merke den Inhalt.
3. Lösche den gefunden Knoten.
4. *Ersetze Inhalt des zu löschenden Knotens durch gemerkten.*

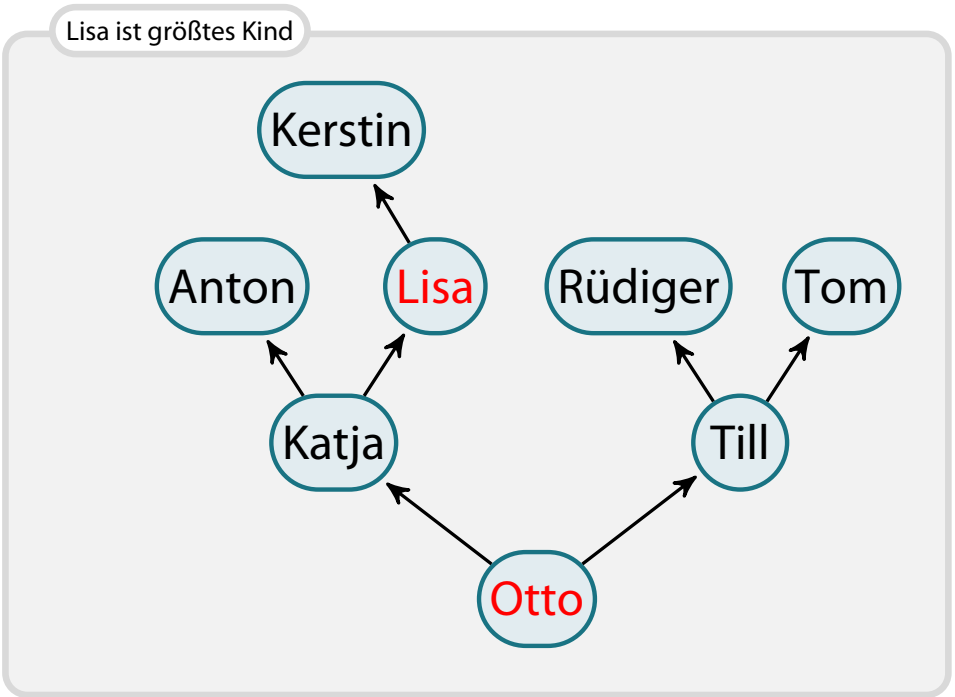


Löschen von *Otto*

Fall 3: Knoten mit zwei Kindern

Löschen eines *Knoten mit zwei Kindern*:

1. *Finde den größten Knoten im linken Teilbaum.*
2. *Merke den Inhalt.*
3. Lösche den gefunden Knoten.
4. Ersetze Inhalt des zu löschenden Knotens durch gemerkten.



Löschen von *Otto*

Fall 3: Knoten mit zwei Kindern

Löschen eines *Knoten mit zwei Kindern*:

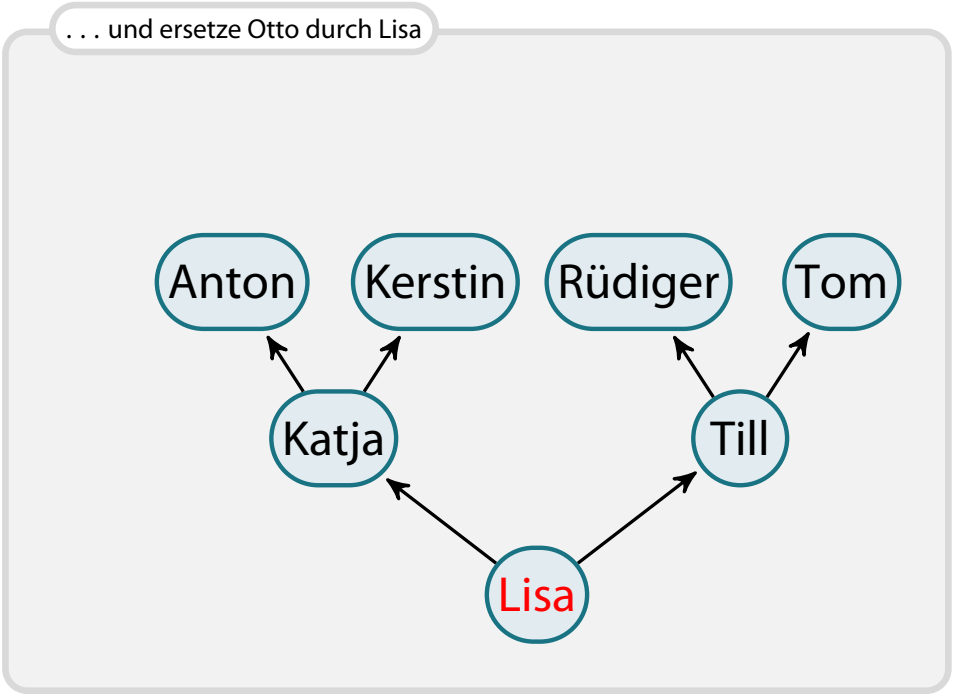
1. Finde den größten Knoten im linken Teilbaum.
2. Merke den Inhalt.
3. Lösche den gefunden Knoten.
4. Ersetze Inhalt des zu löschenden Knotens durch gemerkten.

Löschen von *Otto*

Fall 3: Knoten mit zwei Kindern

Löschen eines *Knoten mit zwei Kindern*:

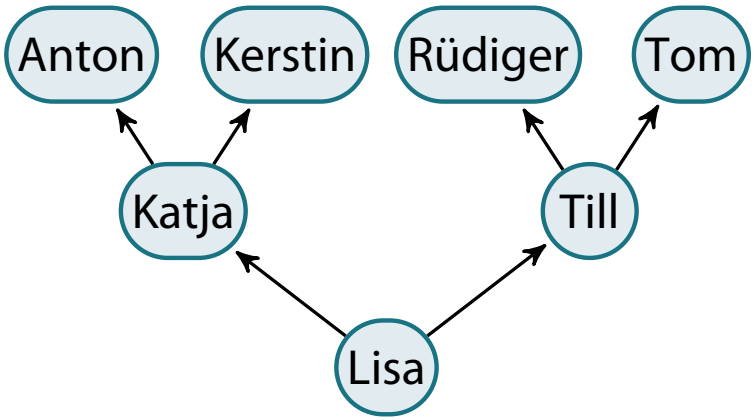
1. Finde den größten Knoten im linken Teilbaum.
2. Merke den Inhalt.
3. Lösche den gefunden Knoten.
4. *Ersetze Inhalt des zu löschenden Knotens durch gemerkten.*



5-Minuten-Aufgabe

Zur Übung

Wie sieht der Suchbaum aus, wenn nacheinander erst *Lisa*, dann *Till* und dann *Kerstin* gelöscht werden?



Vergleich von Listen, Arrays und Suchbäumen

Zeitverbrauch bei n Datenelementen und Baumhöhe h.

Implementation	Suchen	Einfügen	Löschen
sortierter Array	$O(\log n)$	$O(n)$	$O(n)$
unsortierte Liste	$O(n)$	$O(1)$	$O(n)$
Suchbaum	$O(h)$	$O(h)$	$O(h)$

- Offensichtlich ist *die Höhe h* wichtig –
  je kleiner desto besser.
- Ist der Baum *ausgeglichen*, so ist $h = \log_2 n$.
- Ist der Baum *zufällig*, so ist ebenfalls $h = O(\log n)$.
- Werden die Einträge aber *in sortierter Reihenfolge eingefügt* ist sie aber $O(n)$; hier helfen *fortgeschrittene Suchbäume*.

Zusammenfassung

Zentrale Eigenschaften von Suchbäumen

Ein Suchbaum ist ein binärer Baum, in dem für jeden Knoten gilt:

1. Alle Knoten im *linken Unterbaum* sind *kleiner* als der Knoten.
2. Alle Knoten im *rechten Unterbaum* sind *größer oder gleich* dem Knoten.

Die drei zentralen Operationen Suchen, Einfügen und Löschen benötigen Zeit $O(h)$, wobei h die Baumhöhe ist. Sie ist bei zufälligen Bäumen $O(\log n)$, im schlimmsten Fall aber auch $O(n)$.

Zusammenfassung

Suchen in Suchbäumen

Beginnend bei der Wurzel, gehe zum linken Kind, wenn das gesuchte Element kleiner als die Wurzel ist, sonst zum rechten Kind.

Einfügen in Suchbäumen

Suche die Stelle, wo das Element sein müsste, und füge es dort als neues Blatt ein.

Löschen in Suchbäumen

1. Blätter können einfach gelöscht werden.
2. Knoten mit nur einem Kind können durch dieses Kind ersetzt werden.
3. Bei Knoten mit zwei Kindern, finde den größten Knoten im linken Teilbaum, lösche ihn dort und ersetze mit seinem Wert den eigentlich zu löschenden Knoten.