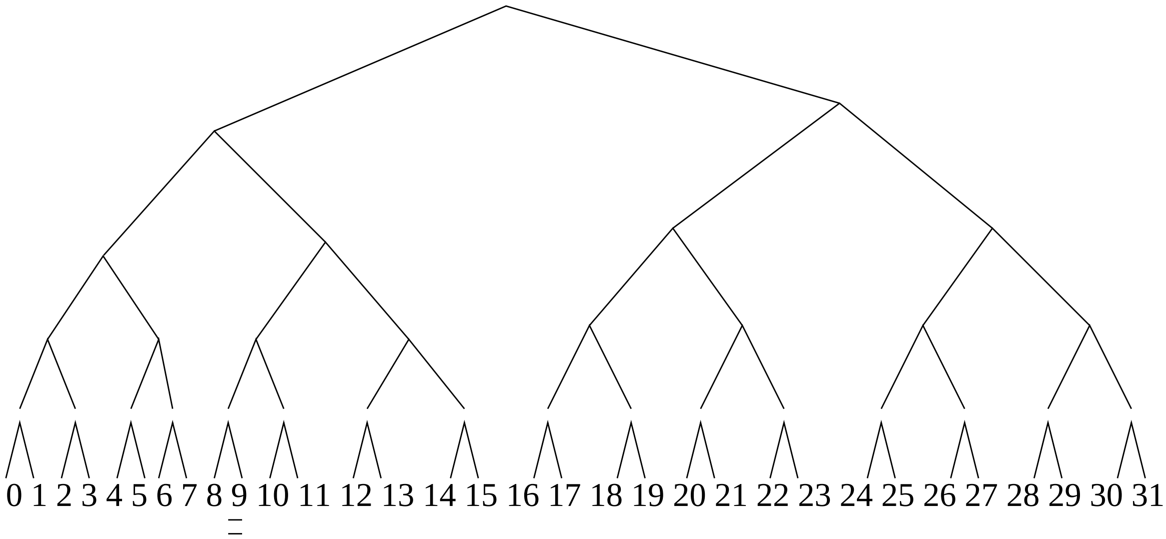
Binärer Suchbaum

B ist ein binärer Suchbaum, wenn

- B ist leer oder
- der linke und der rechte Unterbaum sind binäre Suchbäume
- ist w der Inhalt der Wurzel, so sind alle Elemente im linken Unterbaum kleiner als alle Elemente im rechten Unterbaum gemäß einer Ordnungsrelation.

ein Beispiel



Hier wird die Ordnungsrelation der Elemente ausgenutzt durch die Ordnungsrelation des Baums. Dadurch wird die Suche einfach, das Einfügen und Löschen etwas schwieriger.

Aufwand der Suche

Im schlimmsten Fall ist das gesuchte Element das letzte, das unser Suchverfahren ansieht. Die Länge des Pfades von der Wurzel zum letzten Blatt hängt von der Höhe des Baumes und seiner Breite ab.

- Die Höhe ist 0, wenn der Baum leer ist.
- Die Höhe h(B) ist

  1 + max(h(B.links), h(B.rechts))

Ein binärer Baum B hat die Breite 2.

Die Anzahl der Knoten von B mit h(B)=n ist höchstens

$$m = 2^n - 1.$$

Hat ein Baum m Knoten, so brauchen wir mindestens $logm$ und höchstens m Vergleiche, um ein bestimmtes Element zu finden.

Suchbaum in JAVA

```
import AlgoTools.IO;

/** Klasse Suchbaum: Alle Elemente im linken Unterbaum sind kleiner
 * als die im rechten Unterbaum. Inhalt kann nur vom Typ int sein.
 */

public class Suchbaum extends Baum {

    public Suchbaum () {
        super ();
    }

    public int getIntValue () {
        return ((Integer)inhalt).intValue();
    }
    public String toString () {
        if (empty ())
        return "";
        else return super .toString();
    }
```

```
public static void main (String[] args) {
        Suchbaum baum= new Suchbaum ();
        baum.insert (16);
        baum.insert (10);
        baum.insert (24);
        baum.insert (9);
        baum.insert (14);
        baum.insert (18);
        baum.insert (13);
        baum.insert (15);
        System.out.println (baum.toString ());
        baum.delete (16);
        System.out.println (baum.toString ());
    }
}
```

Ausgabe

java Suchbaum

(((9)10((13)14(15)))16((18)24))

(((9)10((13)14))15((18)24))

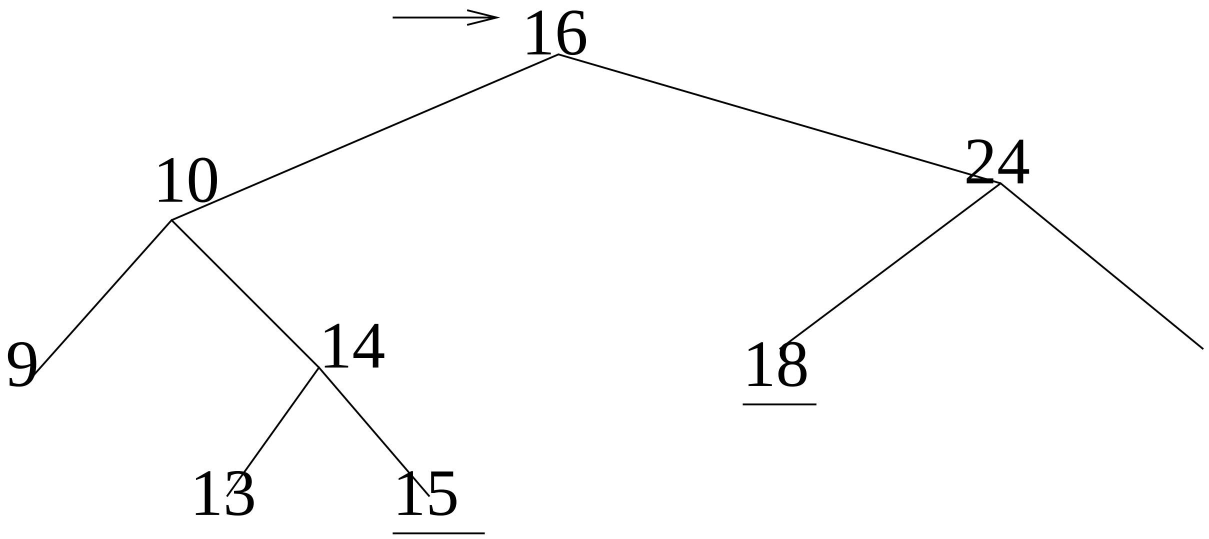
Einfügen

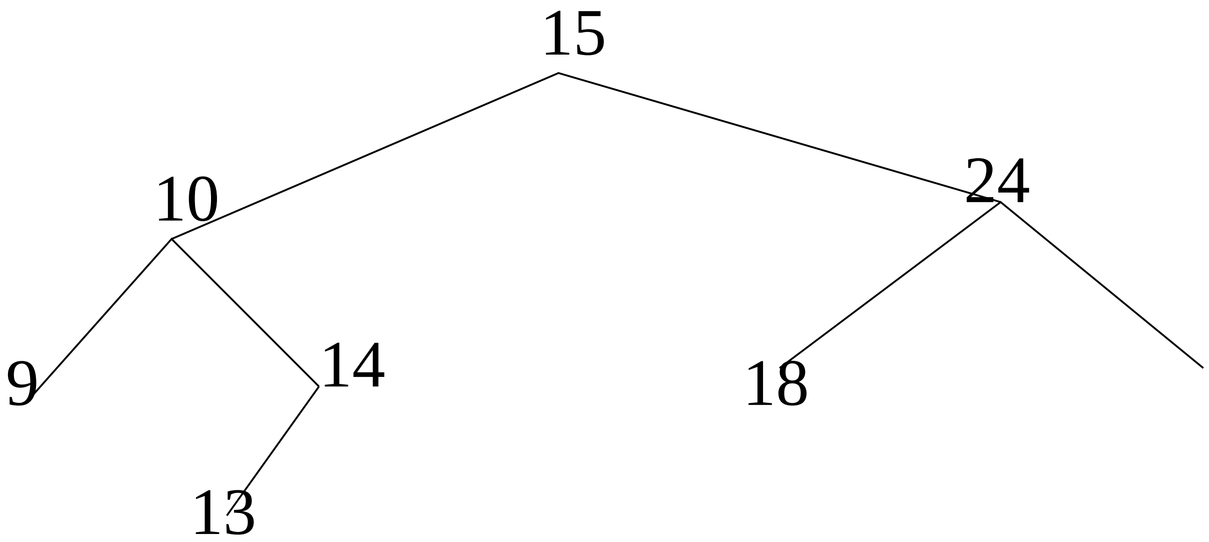
```
public void insert (int neu) {
    if (inhalt == null) {
        inhalt = new Integer (neu);
        links = new Suchbaum ();
        rechts = new Suchbaum ();
    }
    else {
        int intInhalt = getIntValue ();
        if (neu < intInhalt)
        ((Suchbaum)links).insert(neu);
        else
        ((Suchbaum)rechts).insert(neu);
    }
}
```

Finden

```
public Suchbaum find (int x) {
    if (empty ())
    return null;
    int intInhalt = getIntValue ();
    if (intInhalt == x)
    return this ;
    else if
    (x < intInhalt)
    return ((Suchbaum)links).find(intInhalt);
    else
    return ((Suchbaum)rechts).find(intInhalt);
}
```

Löschen





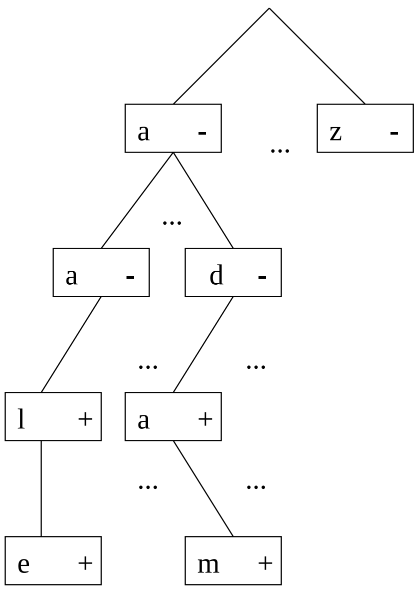
in JAVA

```
public void delete (int lr) {
      Suchbaum stelle= find (lr);
      if (stelle==null)
      return ;
      else if (!stelle.links.empty())
      stelle.inhalt=new Integer
(((Suchbaum)links).linksDelete());
      else if (!stelle.rechts.empty()) {
           stelle.inhalt= stelle.rechts.inhalt;
           stelle.links= stelle.rechts.links;
           stelle.rechts= stelle.rechts.rechts;
      }
      else inhalt=null;
 }

 public int linksDelete () {
      if (rechts.empty ()) {
           int b= getIntValue ();
           delete ( b);
           return b;
      }
      else return ((Suchbaum)rechts).linksDelete();
 }
```

LTree

ist ein Baum, bei dem jeder Knoten ein Zeichen aus einem geordneten Alphabet darstellt und eine Markierung. Ein Wort ergibt sich aus der Konkatenation der Zeichen, die von der Wurzel zu dem Knoten mit positiver Markierung führen.



Realisierung

Breite des Baumes ist nicht (wie beim Binärbaum) vorgegeben. Statt fester Eigenschaften (links, rechts) jetzt: Feld von Nachfolgeknoten. Die Klasse **LTree** hat die folgenden Methoden:

LTree(char c) nimmt einen Buchstaben, markiert ihn und erzeugt das Feld seiner Nachfolger.

insert(String s) fügt ein neues Wort in den Baum ein, indem für jeden Buchstaben der Reihe nach der Konstruktor aufgerufen wird.

remove(String s) entfernt ein Wort aus dem Baum.

hasNext() prüft, ob ein Knoten noch Nachfolger hat.

getIndex(char c) bestimmt den Zahlenwert des Buchstabens.

dump() gibt den Baum aus.

dump(String s) gibt den Baum rekursiv aus, von der Wurzel bis s und dann die Nachfolger von s.

in JAVA

```
public class LTree {
    private LTree[] nachfolger;          // Liste mit Nachfolgern
    protected boolean entry;                         // Markierung
    private static String alphabet =
"?abcdefghijklmnopqrstuvwxyz";                       // Alphabet

    LTree (char c) {
        nachfolger = new LTree[alphabet.length ()];
        entry = false;                         // keine Markierung
    }

    public void insert (String s) {
        if (s.length () == 0) {                      //String leer
            entry = true;                         //Eintrag fertig
        } else {
            char erster = s.charAt (0);
            int pos = getIndex (erster);

            if (nachfolger[pos] == null) {
                                      // evtl. Nachfolger eintragen
                nachfolger[pos] = new LTree (erster);
            }
            nachfolger[pos].insert(s.substring (1));     // Rest
        }
    }
```

```
public boolean remove (String s) {
    if (s.length () == 0) {                         // String leer
        entry = false;          // Hier ist kein Eintrag mehr
        return !hasNext(); // true, wenn keine Nachfolger
    } else {                                  // Erster Buchstabe
        int pos = getIndex (s.charAt (0));
        if (nachfolger[pos] == null) {
                                          // Eintrag nicht im Baum
            return false;
        } else {                        // restliche Buchstaben
            if (nachfolger[pos].remove(s.substring (1)))
{
                nachfolger[pos] = null;           // loeschen
                return (!entry) && (!hasNext());
            } else return false;
        }
    }
}
```

```
public boolean hasNext () {              // Gibt es Nachfolger
        int i = 0;
        while
        (i < nachfolger.length && nachfolger[i] != null)
        i++;
        return i != nachfolger.length;
    }

    public int getIndex (char c) {        // Bestimme Index
        return Math.max
        (0, alphabet.indexOf (Character.toLowerCase (c)));
    }

    public void dump () {                           // Wrapper
        dump ("");
    }
    private void dump (String bisher) {
        if (entry) System.out.println (bisher);
                // Alle existierenden Nachfolger auch abgehen
        for (int i = 0; i < alphabet.length (); i++) {
            if (nachfolger[i] != null)
            nachfolger[i].dump(bisher + alphabet.charAt (i));
        }
    }
}
```

Beispiel

```
import AlgoTools.IO;

public class LTreeBeispiel  {

    public static void main (String[] argv)  {
        int eingabe;              // Entscheidung des Benutzers
        LTree baum = new LTree (' '); // Wurzel des Baumes

        IO.println ("Ich habe einen Baum vorbereitet: ");

        baum.insert ("abraham");
        baum.insert ("adam");
        baum.insert ("alter");
        baum.insert ("debug");
        baum.insert ("dame");
        baum.insert ("adalbert");
        baum.dump ();
```

```
do {
            eingabe = IO.readInt ("Was nun ? (0:Ende
1:Einfuegen "
            + "2:Loeschen) ");
            if (eingabe == 1) {
                baum.insert (IO.readString ("Was? "));
            } else if (eingabe == 2) {
                baum.remove (IO.readString ("Was? "));
            }
            baum.dump ();
        } while (eingabe > 0);

        IO.println ("Auf Wiedersehen!");

    }

 }
```

Ausgabe

Ich habe einen Baum vorbereitet:
abraham
adalbert
adam
alter
dame
debug

Was nun ? (0:Ende 1:Einfuegen 2:Loeschen) 1
Was? doris
abraham
adalbert
adam
alter
dame
debug
doris

Ordnungsrelation

Beim Binärbaum: links, rechts;

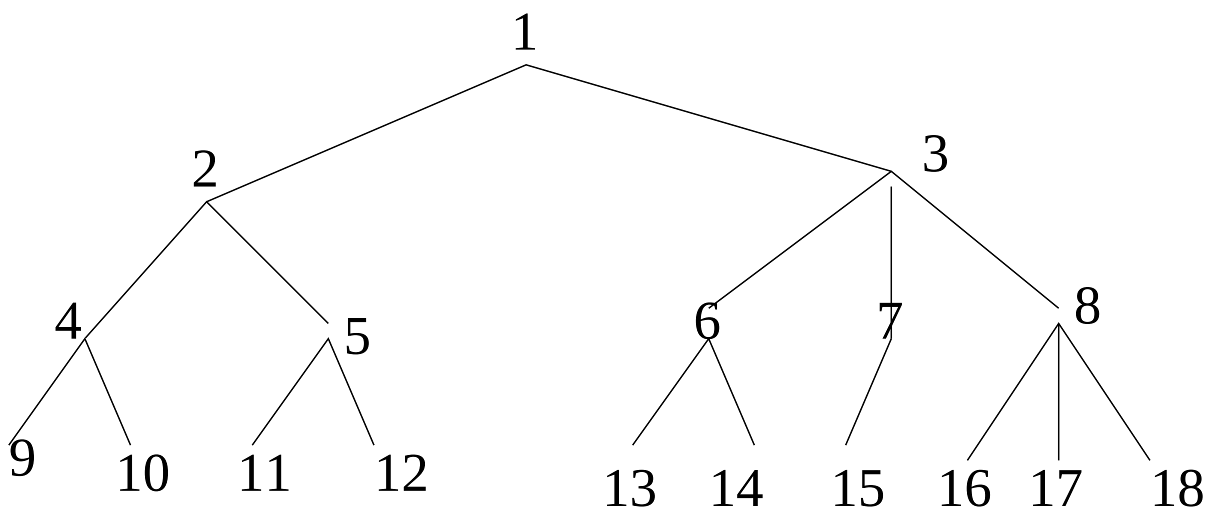
beim binären Suchbaum: Ausnutzen der Ordnungsrelation über den Elementen, so daß alle Elemente im linken Unterbaum kleiner als die im rechten sind;

beim LTree: Ausnutzen der Ordnungsrelation über den Elementen, so daß für jeden Unterbaum Tl, der links von einem anderen Unterbaum Tr ist, gilt, daß alle Elemente von Tl kleiner als die in Tr sind.

Suchstrategien

Rekursive Tiefensuche, iterative Breitensuche

gelingen auch bei ungeordneten Bäumen bzw. erschöpfender Suche.



Breitensuche: 1,2,3,...,18
Tiefensuche in preorder (erst die Wurzel, dann links, dann rechts):
1, 2, 4, 9, 10, 5, 11, 12, 3, 6, 13, 14, 7, 15, 8, 16, 17, 18