Gliederung des Kapitels

3.2 Suchverfahren

~~3.2.1 Suchen in sequentiellen Strukturen~~

~~3.2.2 Bäume und (binäre) Suchbäume~~

~~3.2.3 Optimale Suchbäume~~

~~3.2.4 Balancierte Bäume, AVL-Bäume~~

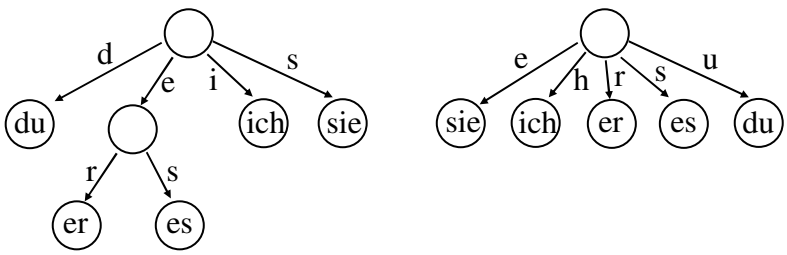
~~3.2.5 B-Bäume~~

3.2.6 Digitale Suchbäume (Tries)

3.2.6 Digitale Suchbäume

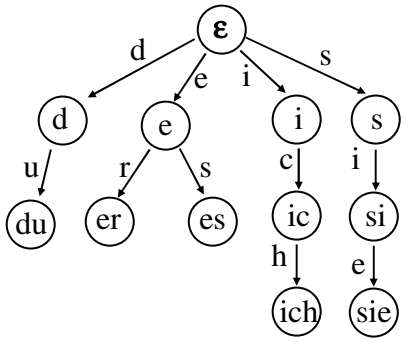
Sind Schlüssel Wörter über einem Alphabet (und das sind sie meistens), so kann man sie zeichenweise (digit per digit, also "digital") lesen und hiermit gleichzeitig einen Baum durchlaufen, bis die restliche Zeichenfolge eindeutig auf den Schlüssel schließen lässt.

Beispiel: Die Schlüssel lauten "ich", "du", "er", "sie", "es".

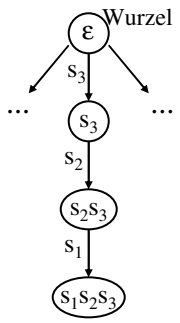


Durchlaufen der Schlüssel von vorne (links) und hinten (rechts).

Baum mit vollständigen Pfaden (von vorne nach hinten durchlaufener Schlüssel):
Die Schlüssel lauten wieder "ich", "du", "er", "sie", "es".

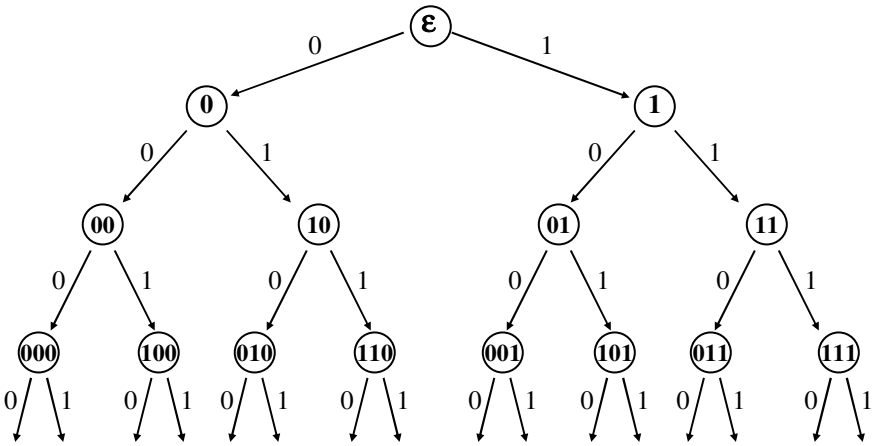


Der Pfad von der Wurzel zu einem Knoten mit dem Schlüssel $s_1s_2...s_k$ ist mit genau diesen Zeichen markiert. Dabei kann man Abkürzungen vornehmen, sofern die weitere Zeichenfolge eindeutig (=eine Kantenfolge ohne Abzweigungen) ist. *Wir lesen hier die Schlüssel von hinten!*

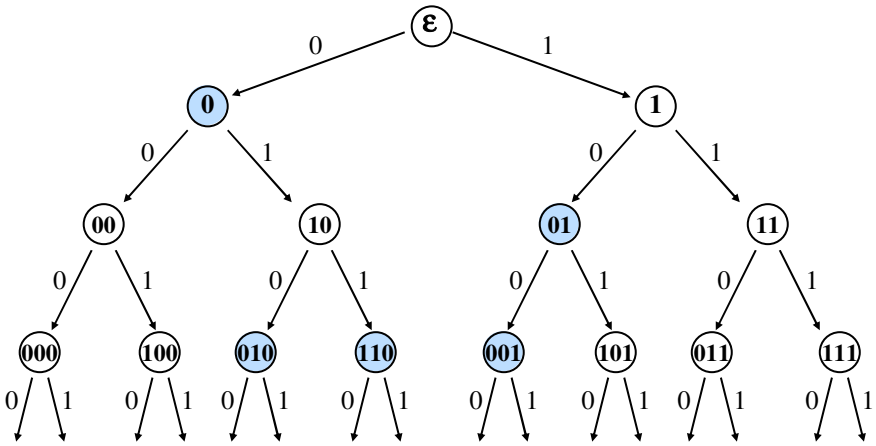


Viele Knoten dienen nur als Verzweigungsknoten zu den Knoten der Schlüssel. Prinzipiell muss man in jedem Knoten mit so vielen Söhnen rechnen, wie es Zeichen im Alphabet gibt. Z.B.: Buchstaben und Ziffern für Bezeichner oder alle Nicht-Steuerzeichen des ASCII-Alphabets.

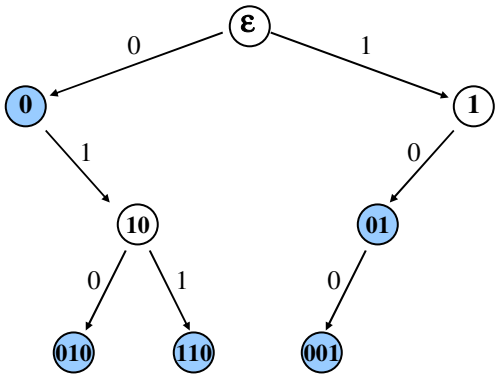
Am Beispiel des Alphabets $\Sigma = \{0, 1\}$ lässt sich dies gut demonstrieren.



Jeder Knoten entspricht der Beschriftung auf dem Pfad von der Wurzel zu ihm. 0 = linker Sohn, 1 = rechter Sohn.

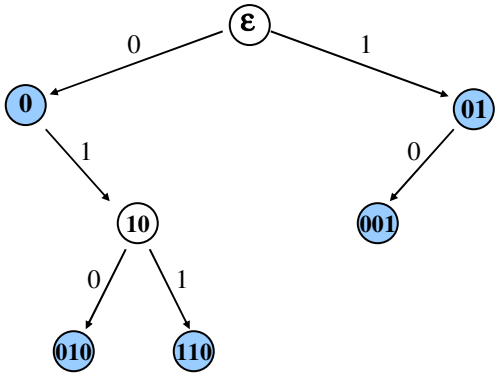


Die Menge {0, 01, 010, 110, 001} im unendlichen Baum.



Die Menge {0, 01, 010, 110, 001} als digitaler Baum.

Ein Schlüssel ist nicht im Baum, wenn es während des zeichenweisen Eingebens keine Fortsetzung mehr gibt (z.B. 11, 00, 101) oder der Schlüssel nur auf einen Verzweigungsknoten führt (z.B. 10).



Etwas verkürzter Baum für die Menge {0, 01, 010, 110, 001}. Beim Suchen und Einfügen muss man dann ab dem aktuell im Knoten stehenden Schlüssel weiter vergleichen.

Digitaler Suchbaum über {0,1}:

Ansatz 1: Gegeben sind Schlüssel als 0-1-Folgen. Für jeden solchen Schlüssel durchlaufe man den 0-1-Baum entsprechend der Binärfolge des Schlüssels und platziere ihn genau in dem zum Schlüssel gehörenden Knoten.

FIND, INSERT und DELETE verhalten sich nun wie bei binären Suchbäumen.

Vorteile dieser Struktur vor allem dann, wenn man auf Maschinenebene programmiert oder die Binärdarstellung von Schlüsseln vorliegt.

Digitaler Suchbaum über {0,1}:

Ansatz 2: Verkürzung der Pfade um lineare Ketten. Gegeben sind Schlüssel als 0-1-Folgen. Für jeden solchen (stets vorwärts- oder stets rückwärts gelesenen) Schlüssel durchlaufe man den 0-1-Baum und platziere den Schlüssel auf den ersten freien Knoten.

Damit dies korrekt funktioniert, muss eine Nebenbedingung gelten. Wie lautet sie?? (Bsp.: füge 0110 und 1110 ein.)

FIND, INSERT und DELETE verhalten sich nun wiederum wie bei binären Suchbäumen, jedoch muss in jedem Knoten auf Gleichheit während des Durchlaufs geprüft werden.

Digitale Suchbäume über beliebigen Alphabeten ergeben sich als nahe liegende Verallgemeinerung.

Mögliche Datenstruktur für digitale Bäume über {0,1}:

```
Maxlaenge: constant Positive := ... ;
type Stelle is (0,1,#);
type Schluesseltyp is array (1..Maxlaenge) of Stelle;

type Knotentyp;
type Baumtyp is access Knotentyp;
type Knotentyp is record
        Schluessel: Schluesseltyp := (others => #);
        L, R: Baumtyp;
        end record;
```

Für eine natürliche Zahl (=Schlüssel) s sei
bit(s,k) = if $s < 2^k$ then k-tes Bit von hinten in der Binärdarstellung von s else # fi.
bit(s,1) ist also die letzte Binärstelle von s, bit(s,2) die vorletzte usw. Vorne wird mit "#" aufgefüllt (nicht mit 0).

Suche / Einfügen / Löschen für Schlüssel s

```
h1, h2: Baumtyp;  b: Natural := Maxlaenge; Code: Schluesseltyp;
begin  h1 := "Verweis auf den digitalen Suchbaum"; h2 := null;
      "wandle s mittels bits in Schluesseltyp um
            und weise dies Code zu";
      while h1 /= null and then h1.Schluessel /= Code loop
            h2 := h1;
            if Code(b) = 0 then h1:=h1.L;
            elsif Code(b) = 1 then hilf1:=h1.R;
            else h1 := null; end if;
            b:= b-1;
      end loop;
      if h1 = null then "s nicht im Baum; h2 verweist auf den Knoten,
            an den ein Knoten mit Schlüssel s angehängt werden kann"
      else "h1 verweist auf Knoten mit dem Schlüssel s" end if;
end;
```