

3. CoMa ÜbungHuffman algorithmus

- zwei 2er Gruppen gesucht
- PA 1,2,3 ab jetzt in ever SVN legen

Huffman algorithmus:

Ziel: Daten komprimieren durch Wechsel der Codewörter  
 → häufige Zeichen bekommen kurze Codewörter  
 → seltene Zeichen —||— längere Codewörter  
 Wechsel von Blockcode zu Präfixcode

formal: Alphabet  $C$  von Zeichen

Codewort  $c \in C \rightarrow g(c) \in \{0,1\}^*$

Code = Gesamtheit aller Codewörter

Beispiel: Datei ababba c da

Zeichen	a	b	c	d
Häufigk.	4	3	1	1

$\Sigma = 9$

(ein) kürzesten Blockcode:

a → 00

b → 01

c → 10

d → 11

anderen Präfixcode:

a → 0

b → 10

c → 110

d → 111

Datei:

000100010100101100  
 9 · 2 = 18 bits

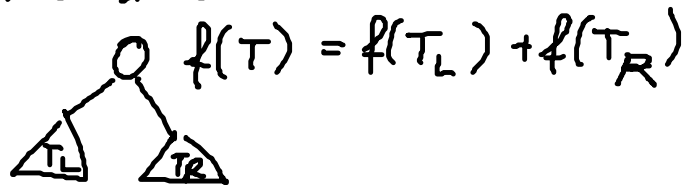
0100101001101110  
 16 bits

Wie finde ich einen guten Präfixcode?

→ Huffman Algorithmus findet optimalen Präfixcode

Huffman algo:

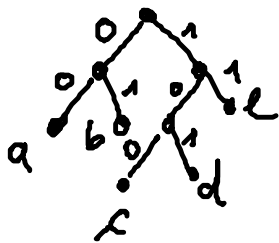
- erstelle Priority Queue für Huffman Bäume
- füge Bäume  $(c)$  mit Häufigkeit  $f(c)$
- solange PQ min. zwei Bäume enthält:
  - entferne die zwei Bäume mit kleinster Häufigkeit
  - verbinde diese zu neuem Baum



- füge neuen Baum in PQ ein
- gib den Baum aus der PQ zurück

Komprimierung:

- Datei einlesen und Häufigkeiten der Zeichen zählen  
Zeichen = Bytes  $\in \{0, \dots, 255\} \Rightarrow$  bspl. int-Array der Länge 256  
 $2^8 - 1$
- Huffmanbaum erstellen aus Zeichen mit pos. Häufigkeit
- Code tabelle erstellen (aus Huffmanbaum)



a → 00  
b → 01  
c → 100  
d → 101  
e → 11

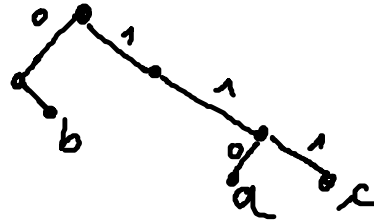
Sinnvoll: wieder Array der Länge 256 vom Typ des Code wortes  
bspl.: `LinkedList<boolean>`  
`boolean[]`  
`String`

- Code tabelle in Ausgabedatei schreiben
- Datei zeichen weise lesen und passende Codewörter schreiben

## De Komprimierung:

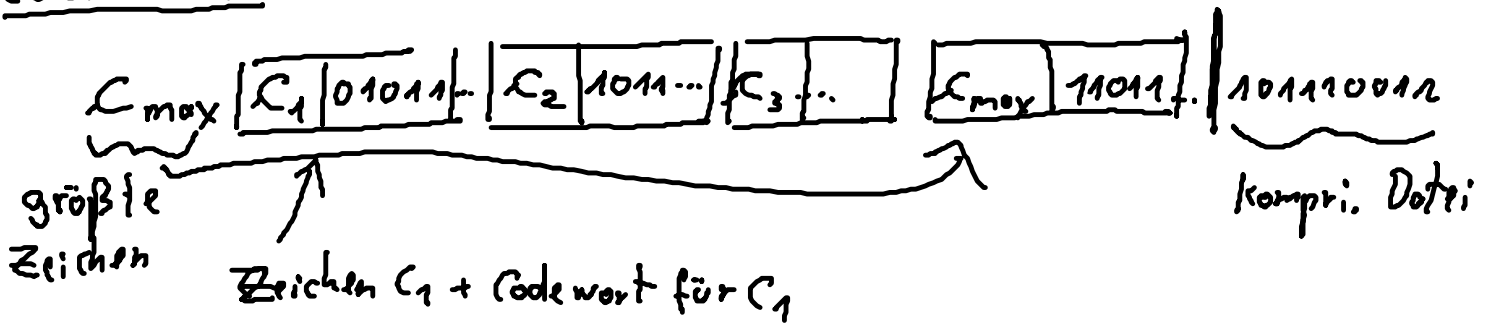
- Codetabelle aus Datei lesen
- Huffmanbaum zur gegeb. Codetabelle erstellen

a → 110  
b → 01  
c → 111  
d :



- bits schrittweise lesen und im Huffmanbaum von wurzel aus nach rechts/links laufen
  - im Blatt: enthaltene Zeichen und springe zurück zur wurzel

## Code tabelle:



- Zeichen sind aufst. sort. in Code tabelle
- größtes vorkomm. Zeichen steht am Anfang ⇒ Codewort für  $C_{max}$  gelesen ⇒ Code tabelle zuende

Wie trennt man Codewort von  $C_{max}$  und kompri. Datei?  
Wie kann man Bits und Bytes schreiben/unterscheiden?

⇒ Klasse BitIO sagt wie viele bits noch zu lesen sind  
BitIO: Klasse/Paket zum Schreiben und Lesen von bytes/bits

(→ siehe javadoc)

- kennt 2 Modi: Bit Mode und Byte Mode

Bit Input File `bf = new BitinOutputFile(filename);`  
Output

- startet im Byte Modus

- Wechsel per `beginBitMode()` / `endBitMode()`

Byte Mode	Bit Mode
<code>readByte()</code> → liefert ein byte als <code>int ∈ {0, ..., 255}</code>	<code>readBit()</code> → liest bit als <code>boolean</code>
<code>writeByte(int)</code> schreibt <code>int ∈ {0, ..., 255}</code> als byte in Datei	<code>writeBit(boolean)</code> schreibt ein bit in die Datei
<del><code>available()</code></del> → keine Information, struktur der Datei nutzen	<code>bitsLeft()</code> → Anzahl an Bits, die noch gelesen werden können

⚠ Methoden im richtigen Modus nutzen

Datei geschrieben/gelesen ⇒ `bf.close()` aufrufen

Unterscheidung zw. letztem Codewort und komprim. Datei:  
Bit Modus beenden und direkt wieder starten

Dateien lesen/schreiben:

File In/Output Stream:

lesen:

```
FileInputStream fis = new FileInputStream(filename);  
BufferedInputStream bis = new BufferedInputStream(fis);
```

```
/**  
 * read file bis.read();  
 *  
 **/
```

} Verarbeitung, solange `read()` bis  
dies -1 liefert

```
bis.close();  
fis.close();
```

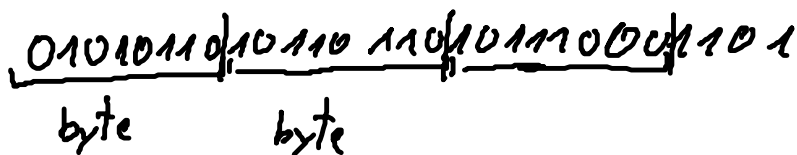
# Schreiben:

```
FileOutputStream fos = new FileOutputStream(filename);  
BufferedOutputStream bos = new BufferedOutputStream(fos);
```

```
/**  
 * write  
 * write file bos.write(int);  
 *  
 */  
  
bos.close();  
fos.close();
```

## Bit IO - Interner:

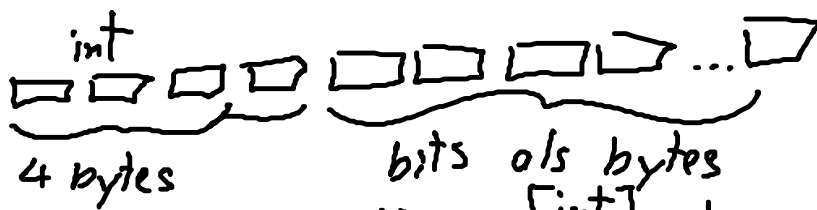
Dateien immer byte-weise:



010101100,

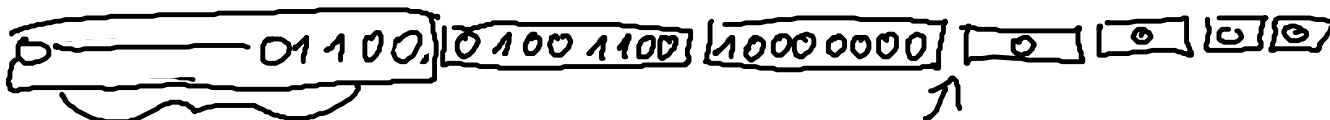
$$\text{int} = \text{Zahl} \in [-2^{31}, 2^{31}-1] = 4 \text{ bytes}$$

BitIO schreibt:



Länge:  $\lceil \frac{\text{int}}{8} \rceil$ , da 0en angehängt

Bsp: Schreibe 010011001000 im bit Modus:



⇒ BitIO kann maximal  $2^{32}-1$  bits in Folge schreiben  
 $2^{32}-1$  bits  $\approx 250$  MB

Huffman: max. Codierungsrate:  $1/8$  bei  $aaaaaa \rightarrow 111111$

besser: mehrere Zeichen als eins kodieren

z.B. Run Length Encoding:  $aaaaaa \rightarrow 6a$   
oft zusammen mit günstiger „Sortierung“ (muss invertierbar)  
sein

$2^{\text{er}}$  Gruppe?