3. CoMa Übung

24.04.

Huffman algorithmus

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

Huffmon 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 abab bac da

Zeichen	a	b	c	d
Häufigk.	4	3	1	1

$\Sigma = 9$

(ein) kürzesten Blockcode:

a → 00
b → 01
c → 10
d → 11

Datei:

00 01 00 01 01 00 10 11 00

9·2 = 18 bits

anderen Präfixcode:

a → 0
b → 10
c → 110
d → 111

0 10 0. 10 10. 0 110 111 0

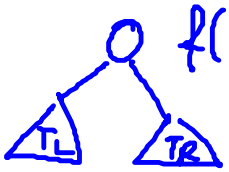
16 bits

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



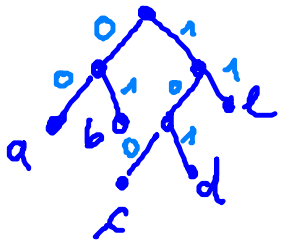
  - füge neuen Baum in PQ ein
- gib den Baum aus der PQ zurück

Komprimierung:

- Datei einlesen und Häufigkeiten der Zeichen zählen

  Zeichen = Bytes ∈ {0, ..., 255} ($2^8 - 1$) ⇒ bspl. int-Array der Länge 256

- 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 Codewortes

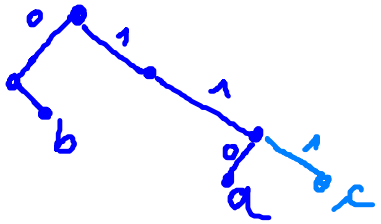
bspl.: LinkedList<boolean>
boolean[]
String

- Code tabelle in Ausgabedatei schreiben
- Datei zeichenweise lesen und passende Codewörter schreiben

Dekomprimierung:

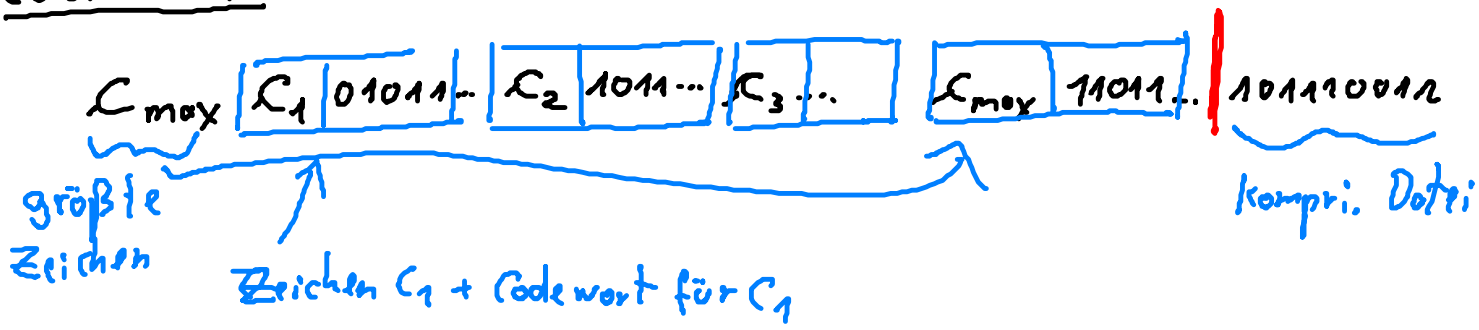
- Codetabelle aus Datei lesen
- Huffmanbaum zur gegeb. Codetabelle erstellen

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

Codetabelle:



- Zeichen sind aufst. sort. in Codetabelle
- größtes vorkomm. Zeichen steht am Anfang => Codewort für c_{max} gelesen => Codetabelle 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

BitInput/OutputFile bf = new BitInput/OutputFile(filename);

- startet im ByteModus
- Wechsel per beginBitMode()/ endBitMode()

ByteMode	BitMode
readByte() → liefert ein byte als int ∈ {0,...,255}	readBit() → liest bit als boolean
writeByte(int) schreibt int ∈ {0,...,255} als byte in Datei	writeBit(boolean) schreibt ein bit in die Datei
~~available()~~ → keine Information, Struktur der Datei nutzen	bitsLeft() → Anzahl an Bits, die noch gelesen werden können

⚠ Methoden im richtigen Modus nutzen

Datei geschrieben/gelesen ⇒ bf.close() aufrufen

Unterscheidung zw. letztem Codewort und kompri. Datei:
BitModus 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();
 *
 **/

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

} Vorarbeitung, solange read() bis dies -1 liefert

Schreiben:

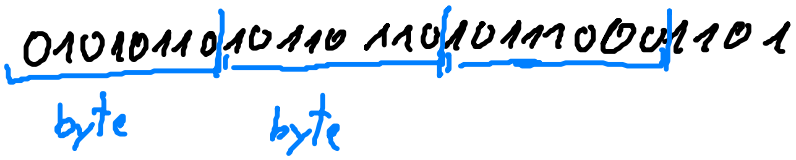
```
FileOutputStream fos = new FileOutputStream(filename);
BufferedOutputStream bos = new BufferedOutputStream(fos);

/**
 *  write file bos.write(int);
 *
 **/

bos.close();
fos.close();
```

Bit IO - Interna:

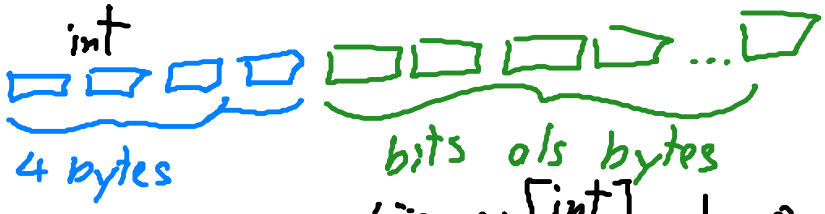
Dateien immer byte-weise:



01010110

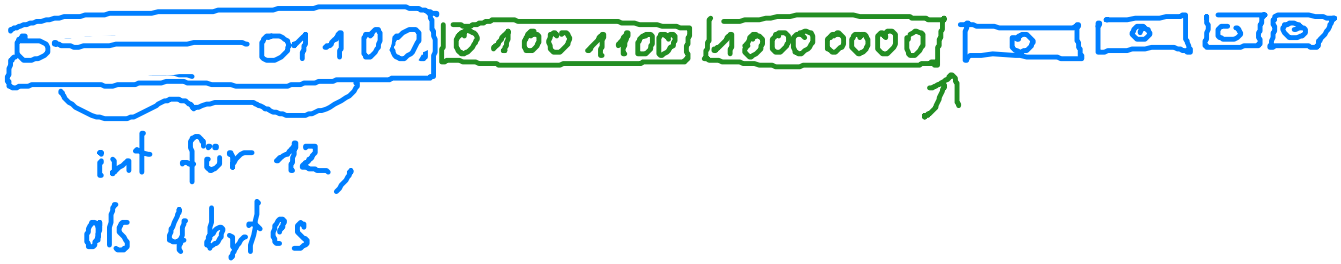
int = Zahl $\in [-2^{31}, 2^{31}-1]$ = 4 bytes

BitIO schreibt:



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

Bsp: Schreibe 0100.1100.1000 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 → 111111

besser: mehrere Zeichen als eins kodieren

z.B. Run Length Encoding: aaaaaa → 6a

oft zusammen mit günstiger „Sortierung" (muss invertierbar sein)

2^er Gruppe?