LZSS und Deflate

Lars Kunert

Seminar Kompressionsalgorithmen
RWTH Aachen

April 18, 2012

Übersicht

LZSS - 1

Verbesserungen gegenüber LZ77:

- Suchbaum
- weniger Redundanz

LZSS - 2

Vorher:

- 4 Bit (Länge) + 12 Bit (Distanz) + 8 Bit (Literal) = 24 Bit

Jetzt bei nicht-Treffer:

- 1 Bit (0) + 8 Bit (Literal) = 9 Bit

Jetzt bei Treffer:

- 1 Bit (1) + 4 Bit (Länge) + 12 Bit (Distanz) = 17 Bit

Deflate - Übersicht

- Kombination aus Huffman und LZSS
- Output besteht aus Blocks $B_1, B_2, ...B_n$
- $B_i = BFINAL, BTYPE, Data$

Deflate - Übersicht

- Kombination aus Huffman und LZSS
- Output besteht aus Blocks $B_1, B_2, ...B_n$
- $B_i = BFINAL, BTYPE, Data$

BTYPE

00 $\Rightarrow$ *unkomprimiert*
01 $\Rightarrow$ *fester Huffman − Code*
10 $\Rightarrow$ *dynamischer Huffman − Code*

Deflate - Unkomprimierter Block

Inhalt:

- 2 Bytes: *LEN*, 2 Bytes: *NLEN*
- *LEN* Bytes: Daten

$LEN < 64$ KiB

Deflate - Komprimierter Block

Was steht in einem komprimierten Block?

Deflate - Komprimierter Block

Was steht in einem komprimierten Block?

- $<Literal>$ oder
- $<Länge, Distanz>$
-

Deflate - Komprimierter Block

Was steht in einem komprimierten Block?

- $<Literal>$ oder
- $<Länge, Distanz>$ oder
- $<end - of - block>$

Deflate - Komprimierter Block

- Huffman-Codierung!

Deflate - Komprimierter Block

- Huffman-Codierung

Symbole

Literal: 0 .. 255
End-Of-Block
Länge: 3 .. 258
Distanz: 1 .. 32 768

Deflate - Komprimierter Block

- Huffman-Codierung

Symbole $\Rightarrow$ 2 Alphabete

Literal: 0 .. 255
End-Of-Block: 256
Länge: 257 .. 512
Distanz: 1 .. 32 768

Deflate - Komprimierter Block

- Huffman-Codierung

Symbole $\Rightarrow$ 2 Alphabete

Literal: 0 .. 255
End-Of-Block: 256
Länge: 257 .. 285 + Extra-Bits
Distanz: 0 .. 31 + Extra-Bits

Deflate - Komprimierter Block, feste Codes

- Für beide Alphabete feste Codes
- Literal/Längen-Alphabet: 7-9 Bits
- Distanz-Alphabet: 5 Bits

Deflate - Komprimierter Block, feste Codes

- Für beide Alphabete feste Codes
- Literal/Längen-Alphabet: 7-9 Bits
- Distanz-Alphabet: 5 Bits

Example

(De-)Kompression

Deflate - Komprimierter Block, feste Codes

Lit Value	Bits	Codes
0 - 143	8	00110000 through 10111111
144 - 255	9	110010000 through 111111111
256 - 279	7	0000000 through 0010111
280 - 287	8	11000000 through 11000111

Deflate - Komprimierter Block, dynamische Codes

- Huffman-Baum übertragen?

Deflate - Komprimierter Block, dynamische Codes

- Huffman-Baum übertragen?

Regeln

- Alle Codes einer Bit-Länge haben lexikographisch aufeinanderfolgende Werte, in der gleichen Reihenfolge wie die Symbole, die sie repräsentieren
- $10 < 000$

Deflate - Komprimierter Block, dynamische Codes

```
code = 0;
bl_count[0] = 0;
for (bits = 1; bits ≤ MAX_BITS; bits++) {
  code = (code + bl_count[bits-1]) << 1;
  next_code[bits] = code;
}
for (n = 0; n ≤ max_code; n++) {
  len = tree[n].Len;
  if (len != 0) {
   tree[n].Code = next_code[len];
   next_code[len]++;
  }
}
```

Deflate - Komprimierter Block, dynamische Codes

- Huffman-Baum per Code-Längen übertragen!

Deflate - Komprimierter Block, dynamische Codes

- Huffman-Baum per Code-Längen übertragen!
- Code-Längen mit Huffman-Code codieren

Deflate - Komprimierter Block, dynamische Codes

Code-Längen-Alphabet

0 - 15: Code-Längen 0 - 15
16: Code-Länge 3 - 6 mal kopieren (+ 2 Extra-Bits)
17: Code-Länge von 0 3 - 10 mal wiederholen (3 Extra-Bits)
18: Code-Länge von 0 11 - 138 mal wiederholen (7 Extra-Bits)

Deflate - Komprimierter Block, dynamische Codes

Code-Längen-Alphabet

0 - 15: Code-Längen 0 - 15
16: Code-Länge 3 - 6 mal kopieren (+ 2 Extra-Bits)
17: Code-Länge von 0 3 - 10 mal wiederholen (3 Extra-Bits)
18: Code-Länge von 0 11 - 138 mal wiederholen (7 Extra-Bits)

Example

8, 16 (11), 16 (10)
”8” 6 mal und 5 mal wiederholen
$\Rightarrow$ 12 mal Code-Länge 8

Deflate - Komprimierter Block, dynamische Codes

- 3 Bits: *BFINAL* und 10
- 5 Bits: *HLIT*, # der Literal/Längen-Codes (257 - 286)
- 5 Bits: *HDIST*, # der Distanz-Codes (1 - 32)
- 4 Bits: *HCLEN*, # der Code-Längen-Codes (4 - 19)

Deflate - Komprimierter Block, dynamische Codes

- 3 Bits: *BFINAL* und 10
- 5 Bits: *HLIT*, # der Literal/Längen-Codes (257 - 286)
- 5 Bits: *HDIST*, # der Distanz-Codes (1 - 32)
- 4 Bits: *HCLEN*, # der Code-Längen-Codes (4 - 19)
- (*HCLEN*) x 3 bits: Code-Längen des Code-Längen-Alphabets

Deflate - Komprimierter Block, dynamische Codes

- 3 Bits: *BFINAL* und 10
- 5 Bits: *HLIT*, # der Literal/Längen-Codes (257 - 286)
- 5 Bits: *HDIST*, # der Distanz-Codes (1 - 32)
- 4 Bits: *HCLEN*, # der Code-Längen-Codes (4 - 19)
- (*HCLEN*) x 3 bits: Code-Längen des Code-Längen-Alphabets
- *HLIT* Code-Längen für: Literal/Längen-Alphabet

Deflate - Komprimierter Block, dynamische Codes

- 3 Bits: *BFINAL* und 10
- 5 Bits: *HLIT*, # der Literal/Längen-Codes (257 - 286)
- 5 Bits: *HDIST*, # der Distanz-Codes (1 - 32)
- 4 Bits: *HCLEN*, # der Code-Längen-Codes (4 - 19)
- (*HCLEN*) x 3 bits: Code-Längen des Code-Längen-Alphabets
- *HLIT* Code-Längen für: Literal/Längen-Alphabet
- *HDIST* Code-Längen für: Distanz-Alphabet

Deflate - Komprimierter Block, dynamische Codes

- 3 Bits: *BFINAL* und 10
- 5 Bits: *HLIT*, # der Literal/Längen-Codes (257 - 286)
- 5 Bits: *HDIST*, # der Distanz-Codes (1 - 32)
- 4 Bits: *HCLEN*, # der Code-Längen-Codes (4 - 19)
- (*HCLEN*) x 3 bits: Code-Längen des Code-Längen-Alphabets
- *HLIT* Code-Längen für: Literal/Längen-Alphabet
- *HDIST* Code-Längen für: Distanz-Alphabet
- Komprimierte Daten

Deflate - Komprimierter Block, dynamische Codes

- 3 Bits: *BFINAL* und 10
- 5 Bits: *HLIT*, # der Literal/Längen-Codes (257 - 286)
- 5 Bits: *HDIST*, # der Distanz-Codes (1 - 32)
- 4 Bits: *HCLEN*, # der Code-Längen-Codes (4 - 19)
- (*HCLEN*) x 3 bits: Code-Längen des Code-Längen-Alphabets
- *HLIT* Code-Längen für: Literal/Längen-Alphabet
- *HDIST* Code-Längen für: Distanz-Alphabet
- Komprimierte Daten
- $<$ End-Of-Block $>$

Deflate - Komprimierter Block, dynamische Codes

- 3 Bits: *BFINAL* und 10
- 5 Bits: *HLIT*, # der Literal/Längen-Codes (257 - 286)
- 5 Bits: *HDIST*, # der Distanz-Codes (1 - 32)
- 4 Bits: *HCLEN*, # der Code-Längen-Codes (4 - 19)
- (*HCLEN*) x 3 bits: Code-Längen des Code-Längen-Alphabets
- *HLIT* Code-Längen für: Literal/Längen-Alphabet
- *HDIST* Code-Längen für: Distanz-Alphabet
- Komprimierte Daten
- < End-Of-Block >

PNG - Übersicht

- Blöcke/Chunks
- Chunk = (Länge, Type, Data, CRC)
-

PNG - Übersicht

- Blöcke/Chunks
- Chunk = (Länge, Type, Data, CRC)
- *IHDR*, *IDAT*, *IEND*

PNG - IDAT Chunk

- Bild zeilenweise durchgehen
-
-
-
-

PNG - IDAT Chunk

- Bild zeilenweise durchgehen
- Pixeldaten in String schreiben (R, G, B)
-
-
-

PNG - IDAT Chunk

- Bild zeilenweise durchgehen
- Pixeldaten in String schreiben (R, G, B)
- String mit Deflate komprimieren
-
-

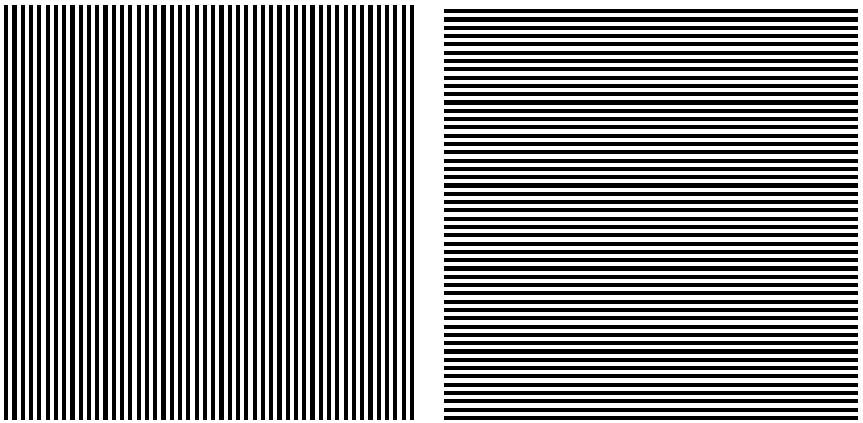
PNG - IDAT Chunk

- Bild zeilenweise durchgehen
- Pixeldaten in String schreiben (R, G, B)
- String mit Deflate komprimieren
- komplette Daten:
  0111 1000 (CINFO: 32K CM: Deflate)
  bb 0 ccccc (FLEVEL, FDICT, FCHECK)
  komprimierter String
  Checksumme (ADLER32)
-

PNG - IDAT Chunk

- Bild zeilenweise durchgehen
- Pixeldaten in String schreiben (*R*, *G*, *B*)
- String mit Deflate komprimieren
- komplette Daten:
  0111 1000 (CINFO: 32K CM: Deflate)
  bb 0 ccccc (FLEVEL, FDICT, FCHECK)
  komprimierter String
  Checksumme (ADLER32)
- evtl. auf mehrere *IDAT*-Chunks aufteilen

PNG - Beispiel



Dateigrößen (100x100 Pixel):

- vertikal: 576 B
- horizontal: 460 B

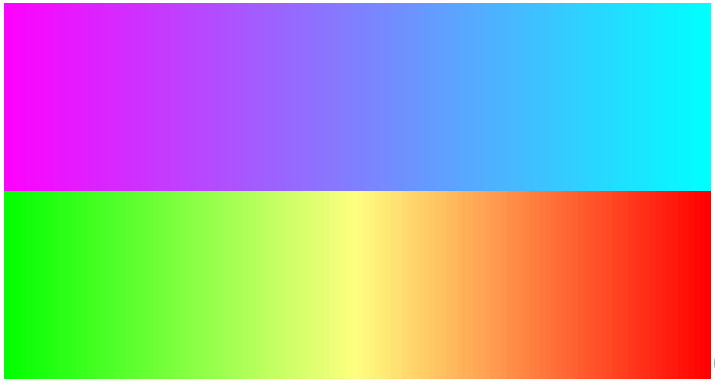
PNG - Kodierer Vergleiche

Bitmap:

- bmp: 9.6 KiB (Bilddaten: 8.5 KiB)

PNG:

-
-
-


(128x68 Pixel)

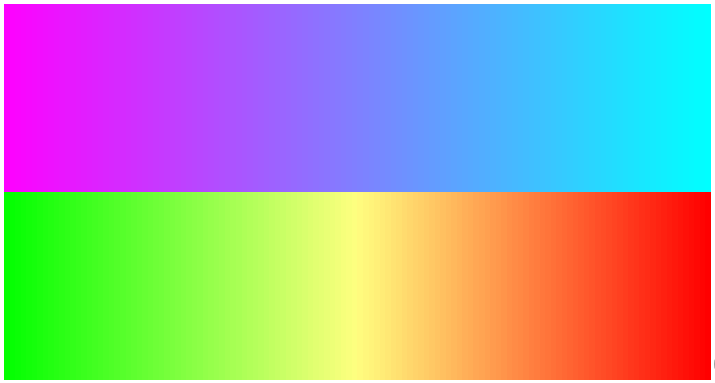
PNG - Kodierer Vergleiche

Bitmap:

- bmp: 9.6 KiB (Bilddaten: 8.5 KiB)

PNG:

- MS Paint: 482 B (IDAT-Chunk: 375 B)
- Paint.NET: 1290 B (IDAT-Chunk: 366 B)
-



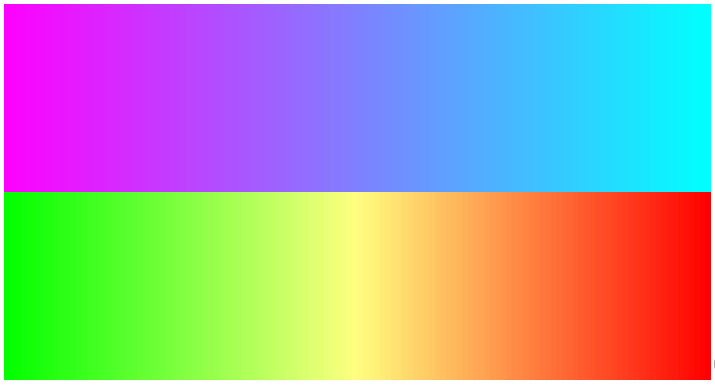
(128x68 Pixel)

PNG - Kodierer Vergleiche

Bitmap:

- bmp: 9.6 KiB (Bilddaten: 8.5 KiB)

PNG:

- MS Paint: 482 B (IDAT-Chunk: 375 B 4.3%)
- Paint.NET: 1290 B (IDAT-Chunk: 366 B 4.2%)
-  (128x68 Pixel)

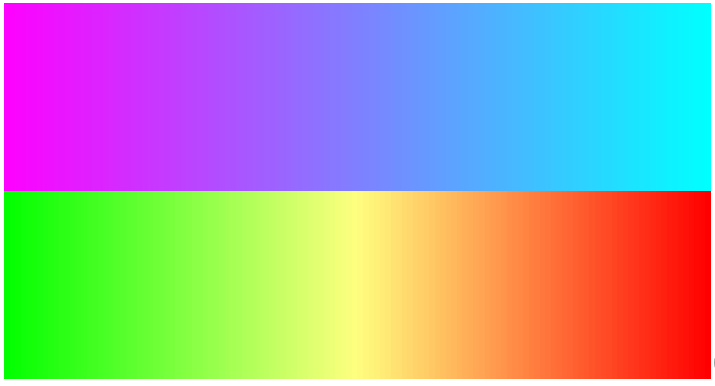
PNG - Kodierer Vergleiche

Bitmap:

- bmp: 9.6 KiB (Bilddaten: 8.5 KiB)

PNG:

- MS Paint: 482 B (IDAT-Chunk: 375 B 4.3%)
- Paint.NET: 1290 B (IDAT-Chunk: 366 B 4.2%)
- optimal mit Vorfilter: 251 B (IDAT-Chunk: 194 B 2.2%)


(128x68 Pixel)

< End-Of-Presentation >

- LZSS
- Deflate
- Anwendung
- nächste Woche: LZMA (1998)