

INTRODUCTION TO MULTIMEDIA SYSTEMS

FALL 2004

EXERCISE #3

CODING AND COMPRESSION

ANDERS BO PEDERSEN

THOMAS A. DALSGAARD

24.09.2004

Theoretical Exercise

Problem 1:

Explain lossless coding.

The name 'lossless' is short for 'without any loss'. Doing lossless coding refers to using compression techniques that will allow one to reconstruct the original data. An example of lossless coding is the RAR compression format. An example of the opposite of lossless coding, lossy, is the JPEG and GIF image formats.

Problem 2:

Explain perceptual coding.

The perceptual coding technique uses the human ability to 'perceive' to model a compression scheme. So the limits of the human senses are here used as an advantage to compress data. This scheme is used widely in audio and video encoding. An example is the MPEG encoders.

Problem 3:

Explain linear predictive coding.

Linear predictive coding encodes uses a selection of perceptual features in the signal and uses a sound synthesizer in the decoding process. The predictive aspect is using the differences in successive quantified signal bits instead of the actual value of the bit itself.

Problem 4:

How would the following sequence be run-length encoded and what percentage smaller is the run-length encoded data?

sequence: 111112223333312222221111111333333333

Runlength code:

1,5 2,3 3,5 1,1 2,2 1,7 3,9

Here 9 is the largest number and corresponds to 1001 in binary numbers. Thus the codeword will have length 4 and the resulting signal will look like the following.

00010101 00100011 00110101 00010001 00100010 00010111 00111001

Of course without the spaces (only here for readability). This contains $7 \cdot 8 = 56$ bits or 7 bytes and should be compared to 36 bytes. The resulting bits is thus only $7/36 = 19,44\%$ of the original represented number of bits.

Problem 5:

Suppose all odd symbols in the previous problem are replaced by 1, and all even symbols are replaced by 0. Is there a further compressed coding which can be defined and what percentage smaller is this scheme relative to the original binary sequence?

Now we have a string again with the data type char which looks like this:

```
111110001111110000001111111111111111
```

The largest number of occurrence is 16 (10000) which gives a codeword of length 5. By assuming that the receiver doesn't know if the first char is a 1 or 0 we would send this information as the first bit send. Then the sequence of numbers that should be send is:

```
5 3 6 6 16
```

In binary form starting with 1:

```
1 00101 00011 00110 00110 10000
```

Again without the spaces. This contains 26 bits and is only $26/(36*8) = 9,03\%$ of the original bits.

Problem 6:

Calculate the coding efficiency of the following code:

Symbols a,b,c,d,e

$P(a) = .5$

$P(b) = .4$

$P(c) = .05$

$P(d) = .03$

$P(e) = .02$

code for a = 00

code for b = 01

code for c = 11

code for d = 100

code for e = 101

Entropy:

```
>> -(0.5*log2(0.5)+0.4*log2(0.4)+0.05*log2(0.05)+0.03*log2(0.03)+0.02*log2(0.02))
```

ans =

```
1.5095
```

Average number of bits:

```
>> 0.5*2+0.4*2+0.05*2+0.03*3+0.02*3
```

ans =

```
2.0500
```

Efficiency:

```
>> 1.5095/2.0500
```

ans =

0.7363

Problem 7:

Calculate the coding efficiency of the following code:

Symbols a,b,c,d,e

P(a) = .3

P(b) = .2

P(c) = .2

P(d) = .2

P(e) = .1

code for a = 00

code for b = 01

code for c = 11

code for d = 100

code for e = 101

Calculate the coding efficiency of the following code:

Entropy:

```
>> -(0.3*log2(0.3)+0.2*log2(0.2)+0.2*log2(0.2)+0.2*log2(0.2)+0.1*log2(0.1))
```

ans =

2.2464

Average number of bits:

```
>> 0.3*2+0.2*2+0.2*2+0.2*3+0.1*3
```

ans =

2.3000

Efficiency:

```
>> 2.2464 /2.3000
```

ans =

0.9767

Problem 8:

Calculate the coding efficiency of the following code:

Symbols a,b,c,d,e all have probabilities .2

code for a = 00

code for b = 01

code for c = 11

code for d = 100

code for e = 101

Entropy:

```
>> -(0.2*log2(0.2)+0.2*log2(0.2)+0.2*log2(0.2)+0.2*log2(0.2)+0.2*log2(0.2))
```

ans =

2.3219

Average number of bits:

```
>> 0.2*2+0.2*2+0.2*2+0.2*3+0.2*3
```

ans =

2.4000

Efficiency:

```
>> 2.3219/2.4
```

ans =

0.9675

Practical Exercise

Exercise 1:

Create your own compression/decompression program in java. The program should be based on either Lempel-Ziv-Welch (LZW) coding or Huffman coding.

Encode and decode the BW-image and the LaTeX-file with your RLE program. The decoded files should be a exact copy of the original. Use the CompareFiles.java program to check it.

A well-known example of a program performing lossless compression is WinZip. Compare the compression ratio using WinZip with that of your program.

Calculate the entropy of the two files. Comment the results.

Intro:

This group decided to implement the Huffman Algorithm. Furthermore we decided to use Java since the programming skills in the group are limited to this language.

The group had no experience in simple tasks such as reading and writing files using Java or in manually producing bitstreams so the task at hand was new to both group members.

Analysis:

First thing planned and executed was research about Huffman related topics such as binary nodes and Huffman nodetrees. Online resources sufficed our needs in regards to this (see links below).

Another important topic was common 'bits and bytes' theory. The group knows this area and the bit operators well in advance but had no real experience in manual conversion between the primitive datatypes of Java and Strings. So a little effort was used on 'brushing up' this area.

Optimal merge:

<http://www.darkridge.com/~jpr5/archive/dads/HTML/optimalMerge.html>

Suns compresison tools:

<http://java.sun.com/developer/technicalArticles/Programming/compression/>

Bit operators:

<http://mindprod.com/jgloss/binary.html>

Implementation:

df

Results:

ficianc

Conclution:

ficianc

