**ICA03 – LZW – Gruppe 7**

**1.2.1 - A**

Dekoding av: 2,3,3,1,3,4,5,10,11,6,10,1

**Motatt - Table - Dekodet til - Ny entry i table**

2 2 B

3 =4 C BC

3 =5 C CC

1 =6 A CA

3 =7 C AC

4 =8 BC CB

5 =9 CC BCC

10 =10 CCC CCC

11 =11 CCCC CCCC

6 =12 CA CCCCC

10 =13 CCC ACCC

1 =14 A CCCA

**Det gir oss følgende table hos mottaker:**

[1] A

[2] B

[3] C

[4] BC

[5] CC

[6] CA

[7] AC

[8] CB

[9] BCC

[10] CCC

[11] CCCC

[12] CCCCC

[13] ACCC

[14] CCCA

**1.2.1 - B**

Se oppg. 1.2.1 (E)

**1.2.1 – C**

Antatt at både hvert symbol (ABC) og tall i lzw-sekvensen

består av 4 bits (hexadecimal / decimal)

Ukodet melding (22\*4): 88bits uten lzw

LZW sekvens (12\*4) = 48bits med lzw

En besparelse på 40 bits gir 40/88\*100 =

45,45% reduksjon i meldingens størrelse ved bruk av lzw i dette tilfelle.

**1.2.1 - D**

Huffman kode: Beregning av sannsynlighet

for hvert symbol:

A:3 / 22 = 13.63%

B:2 / 22 = 09.09%

C: 17/22 = 77.27%

Utregning av kode:

S{(A-13) (B-09) (C-72)}

S{(C-72) (A\*B-22)

S{C / A\*B - 1)

Dette gir oss følgende huffman kode:

C = 1

A = 01

B = 00

Melding kodet om med huffman:

Cx17 \* 1 = 17bits

Ax3 \* 2 = 6bits

Bx2 \* 2 = 4bits

Huffman kodet melding = 27bits

Dette gir en besparelse på 88-27 = 61bits

61/88 = 69% besparelse

Da jeg er usikker på om den første antagelsen er riktig (at både lzw sekvens og symbol er 4 bits hver) er det vanskelig å si noe konkret om tallene, men det er allikevel åpenbart at huffman-kode vil være et bedre valg i dette tilfellet. Dette antas å bunne ut i meldingens lengde: LZW algoritmen kommer først til sin rett når den har en tabell full av kombinasjoner av symboler som den kode dekode fra. I dette tilfellet er teksten så liten at gjentatte symboler ikke forekommer så mye, og da mister LZW algoritmen en stor fordel. En kan da anta at dersom meldingens lengde økes konstant, vil man nå et punkt der LZW- algoritmen oppnår større komprimeringsgrad enn huffman-koding, selv om dette ikke er tilfelle her.

**1.2.1 – E**

<https://github.com/ErikHD/IS-105_2016_Gruppe07/blob/master/uke%205/lzw.py>

**1.2.1 – F**

Hamlet – 180 kB komprimert til 87kB = 93kB besparelse = 51.6% komprimeringsgrad

Shakespeare – 5459 kB komprimert til 128kB = 5331kB besparelse = 97.6% komprimeringsgrad

Usikker på om LZW-algoritmen fungerer helt korrekt for shakespeare, men moment fra 1.2.1 D er allikevel relevant, dvs meldingens lengde. En liten melding (hamlet) vil ikke kunne oppnå optimal komprimering ved LZW, mens en større melding (shakespeare) oppnår langt større komperingsgrad da det er langt flere forekomster av gjentatte (kombinasjoner av) symboler. Det tolkes dithen at desto større denne meldingen er, desto bedre komprimeringsgrad vil LZW gi.