Komprimering av eBooks enligt Open eBook Publication Structure

(HS-IDA-EA-03-103)

Mattias Borén (a00matbo@ida.his.se)

Institutionen för datavetenskap
Högskolan i Skövde, Box 408
S-54128 Skövde, SWEDEN

Examensarbete på programmet för systemprogrammering under vårterminen 2003.

Handledare: Henrik Grimm

Komprimering av eBooks enligt Open eBook Publication Structure

Examensrapport inlämnad av Mattias Borén till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för Datavetenskap.

2003-06-06

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: ____

Komprimering av eBooks enligt Open eBook Publication Structure

Mattias Borén (a00matbo@ida.his.se)

Sammanfattning

I denna rapport undersöks hur elektroniska böcker (eBooks) enligt standarden Open eBook Publication Structure kan komprimeras på ett effektivt sätt. Anledningen till att området valts är att en välanvänd standard för eBooks inte finns för tillfället. Standarden är relativt ny och senaste versionen av specifikationen har kommit ut under 2002. Med i utvecklingen av standarden finns många stora företag som exempelvis Adobe Systems Inc. och Palm Digital Media. En litterär undersökning för passande komprimeringsalgoritmer har gjorts och två algoritmer för komprimering av eBooks har efter undersökningen utvecklats. Tester och slutsatser angående komprimeringseffektiviteten hos komprimeringsalgoritmerna finns också dokumenterade i rapporten.

Nyckelord: Komprimering, eBook, elektroniska böcker

Innehållsförteckning

1 Introduktion ... 1
2 Bakgrund ... 3
2.1 eBooks ... 3
2.1.1 Open eBook Forum och Open eBook Publication Structure ... 3
2.1.2 Markupspråk ... 4
2.1.2.1 SGML ... 4
2.1.2.2 HTML ... 4
2.1.2.3 XML ... 5
2.1.2.4 XHTML ... 6
2.2 Komprimeringsalgoritmer ... 6
2.2.1 Huffmankomprimering ... 6
2.2.2 Lempel-Ziv komprimering ... 7
2.2.2.1 LZ77 ... 7
2.2.2.2 LZ78 ... 7
2.2.3 Aritmetisk komprimering ... 8
2.2.4 Prediction by Partial Match (PPM) ... 9
2.3 Komprimeringsverktyg ... 10
2.3.1 XMill ... 10
2.3.2 XMLPPM ... 10
3 Problemdefinition ... 11
3.1 Problemspecificering ... 11
3.2 Förväntat resultat ... 11
4 Metod ... 12
4.1 Analys av OeBPS ... 12
4.2 Analys av eBooks enligt OeBPS ... 13
4.3 Val av algoritmer för implementation ... 13
5 Genomförande ... 14
5.1 Implementering av primitiv komprimeringsalgoritm ... 14
5.1.2 Komprimering ... 14
5.1.3 Dekomprimering ... 15
5.2 Implementering av Huffmankomprimeringsalgoritm ... 16
5.2.1 Huffmankodning i detalj ... 16

5.2.2 Komprimering 17
5.2.3 Dekomprimering 19
6 Resultat 20
7 Sammanfattning 24
7.1 Diskussion 24
7.2 Framtida arbete 24
Referenser 26

1 Introduktion

I detta kapitel ges en kort introduktion till de två huvudämnen inom arbetet, eBooks och komprimering. Mer detaljerat beskrivs dessa två ämnen i senare kapitel.

eBooks

Elektronisk lagring av böcker (eBooks) har många fördelar jämfört med vanliga böcker av papper. En av dessa fördelar är att det fysiska utrymmet som krävs för att lagra böcker i denna form är mycket mindre än utrymmet som krävs för att lagra pappersböcker. Betrakta exempelvis ett bibliotek som kräver stora lokaler för att lagra pappersböcker. Samma böcker om de fanns som eBooks skulle kräva mycket mindre lokaler genom att de kunde ha lagrats på ett antal större lagringsmedier. Ytterliggare en fördel med eBooks jämfört med pappersböcker är att kostnaderna för att trycka böcker försvinner och de materiella kostnaderna för papper försvinner när böckerna lagras i form av eBooks. Kostnaderna för att transportera böckerna minskar också markant eftersom det istället för att kostsamt transportera pappersböcker till konsumenterna går att använda elektronisk överföring för att skicka böckerna i form av eBooks.

En av dom första stora författarna att publicera ett verk som enbart eBook var Stephen King, som i mars år 2000 publicerade sin korta roman ”Riding the Bullet” (King, 2000), vilken inte förrän senare fanns att köpa i annan form än som eBook. Boken såldes av vissa försäljare för $2.50, medan andra gav ut den gratis under några få dagar. Efterfrågan av boken blev över förväntan och efter 24 timmar hade 400 000 beställningar av boken mottagits (Doreen Carvajal, 2000). 1998 kom de första externa eBook-läsarna, Nuvomedias Rocket eBook och SoftBook (personlig kontakt, Andrea Lindgren, Technical Support Gemstar eBook Group Ltd, 28 februari, 2003). De externa eBook-läsarna skiljer sig från handdatorer, för vilka det också finns mjukvara för att läsa eBooks till, eftersom de enbart är utvecklade för att avkoda och presentera eBooks.

Komprimering

Användningen av komprimering började redan innan de första datorerna fanns att tillgå. Brittiska flottan använde i slutet av 1800-talet av ett kommunikationssystem, vilket bestod av ett antal hytter som stod på toppen av kullar från London till militära baser vid kusten. Hytterna stod med 5 mils mellanrum och hade sex stycken fönsterluckor som kunde ses från närliggande hytter. Eftersom hytterna hade sex stycken fönsterluckor kunde 64 (2^6) olika meddelandet skickas, vilket efter att ha reserverat en kombination för varje bokstav lämnade ytterliggare kombinationer oanvända. Dessa kombinationer användes för vanligt förekommande ord såsom ”West”, ”and” och ”the”. Meddelanden skickades från London genom att fönsterluckorna justerades på den första hytten och meddelandet skickades vidare genom att operatörer vid varje hytt ändrade luckorna i sin hytt till samma kombination av öppna och stängda luckor som föregående hytt i ledet. På detta sätt kunde meddelande snabbt skickades flera mil på bara några minuter. Användandet av de överblivna kombinationerna av luckorna för att representera vanligt förekommande ord är en tidig användning av komprimering. Detta eftersom användningen av de extra kombinationerna av luckor gjorde att genomsnittslängden på meddelanden blev kortare än om militären skulle ha använt sig av enbart de kombinationer som motsvarade en bokstav (Bell, Cleary & Witten, 1990).

Ytterliggare en tidig användning av komprimering gjordes av Samuel Morse under 1930-talet när han utvecklade Morsekodning. Morse använde sig av det faktum att inom engelskan är olika bokstäver inte lika vanligt förekommande. Exempelvis är ett ”y” inte lika vanligt förekommande som ett ”e”. För att koda bokstäver så att de skulle gå att skicka via en telegraf använde Morse sig av kombinationer av en kort och en lång signal. Morse tilldelade bokstäver som var vanligt förekommande korta kombinationer av signaler medan de mindre vanligt förekommande fick längre signalkombinationer, se figur 1. Det visade sig att genom att använda Morsekodning minskade genomsnittslängden per tecken till 8.5 signaler jämfört med att kräva 11 signaler om sannolikheten för bokstäver i engelska språket inte betraktades (Bell, Cleary & Witten, 1990).

A ·_ M __ Y ·_·_
B _··· N _· Z __··
C _·_· O ___ 1 ·____
D _·· P ·__· 2 ··___
E · Q __·_ 3 ···__
F ··_· R ·_· 4 ····_
G __· S ··· 5 ·····
H ···· T _ 6 _····
I ·· U ··_ 7 __···
J ·___ V ···_ 8 ___··
K _·_ W ·__ 9 ____·
L ·_·· X _··_ 0 _____

Figur 1 : Internationell Morsekod

Avsikten med denna rapport är att undersöka hur den publikationsstruktur och tillhörande textdokument, vilka eBooks som följer Open eBook-standarden innehåller, kan komprimeras på ett så bra sätt som möjligt med avseende på komprimeringseffektivitet.

De format som används för lagring av text och publikationsstruktur är XHTML och XML. I arbetet kommer befintliga komprimeringsverktyg för XML och XHTML att användas. Tester med de olika verktygen samt de algoritmer som implementerats kommer leda fram till en slutsats om vilket verktyg som är effektivast för komprimering av textdokumenten i eBooks. Rapporten innehåller beskrivningar av de komprimeringsalgoritmer som används av komprimeringsverktygen.

2 Bakgrund

För tillfället utvecklar Open eBook Forum (OeBF) en standard för lagring, struktur och presentation av böcker lagrade i elektronisk form, så kallade eBooks. För att kunna läsa en eBook behövs ett läsningssystem. Ett *läsningssystem* är enligt Open eBook Publication Structure Specification (Open eBook Forum, 2002) (OeBPS), vilket är den senast utgivna specifikationen från OeBF, en kombination av hårdvara och/eller mjukvara som accepterar böcker som följer deras standard. Det som nämns i OeBPS (Open eBook Forum, 2002) om komprimering i läsningssystemen är :

> "Reading Systems may include additional processing functions beyond the scope of this specification, such as compression, indexing, encryption, rights management, and distribution."

Alltså får läsningssystemen innehålla ytterligare processfunktioner, men den nuvarande OeBF-standarden ger inga riktlinjer för vilka sorters komprimering som kan vara lämplig för de okomprimerade format som stöds av standarden.

I detta kapitel kommer först detaljer om eBooks, OeBF och filformat som används av OeBPS att tas upp. Detta följs upp med detaljer angående välkända komprimeringsalgoritmer för text och kapitlet avslutas med en beskrivning av några komprimeringsverktyg för XML.

2.1 eBooks

eBooks är en elektroniskt lagrad bok. Möjligheterna att lagra böcker i digital form har funnits länge men det är först på senare år som lagringsformen eBooks har börjat användas för publicering av böcker. eBooks innehåller inte enbart text utan kan också innehålla bilder, animationer, ljudsekvenser, med mera.

Det finns många olika standarder för lagring av eBooks, många av de företag som tillverkar en eBook-läsare har sin egen standard för vilka böcker som kan läsas av den. Detta är väldigt opraktiskt om någon vill publicera en eBook eftersom publiceringen då måste göras i många olika format för att alla ska kunna läsa den. Open eBook Forum är en organisation som startades för att ta fram standarder för eBooks. En av de standarder som tagits fram är OeBPS som innehåller riktlinjer för hur eBooks ska struktureras vid publicering.

I det fortsatta arbetet kommer ordet eBooks mena elektroniska böcker som följer OeBPS.

2.1.1 Open eBook Forum och Open eBook Publication Structure

Open eBook Forum är en organisation skapad för att utveckla och främja standarder för elektroniska böcker. Organisationen har givet ut en specifikation som anger bland annat hur publicering av eBooks ska utföras och vilka filformat som publikationsstandarden bygger på.

Organisationen består av företag och privatpersoner som har anknytning till elektronisk publicering av böcker. Stora företag som exempelvis Adobe Systems Inc. och Palm Digital Media är med i organisationen.

2.1.2 Markupspråk

Markupspråk (eng. Markup languages) används för att strukturera upp och formatera vanlig text. Dessa språk beskriver (Burnard, 1995) :

- vilken typ av strukturering och formatering som är tillåten,
- var formateringen och struktureringen är tillåten,
- vilken typ av strukturering och formatering som krävs,
- hur formatering skiljer sig från vanlig text,
- vad struktureringen och formateringen betyder.

2.1.2.1 SGML

SGML (Standard Generalized Markup Language) är en standard som använder sig av så kallade taggar för att skilja struktur och vanlig text. Taggar representeras i text genom att de har tecknet ”<” före och tecknet ”>” efter, exempelvis ”<smak>”. Enligt Burnards (1995) så uppfyller SGML alla kraven förutom att ge en tolkning på vad struktureringen och formateringen betyder. För att tolka betydelsen av SGML krävs ytterliggare semantisk information. Med semantisk information menas information om informationen, d.v.s. vad de olika taggarna som används betyder.

2.1.2.2 HTML

HTML (Hypertext Markup Language) är en standard som används framför allt på Internet för att publicera text, bilder, med mera. HTML baseras på SGML men tillför den semantiska information som krävs för att kunna tolka struktureringen och formateringen. Exempel på tolkning av struktur visas i tabell 1.

HTML:	Tolkad HTML :
Winter’s Heart <u>Robert Jordan</u>	**Winter’s Heart** Robert Jordan

Tabell 1 : HTML-kod och tolkad HTML-kod

De formateringstaggar från HTML-standarden som används i exemplet i tabell 1 är ”b”, ”u” och ”br”. Inom HTML tolkas ”b” som fetstil, ”u” som understruket och ”br” som ny rad. Eftersom HTML bygger på SGML används tecknen ”<” och ”>” för att kunna skilja formateringstaggar från vanlig text.

2.1.2.3 XML

XML (Extensible Markup Language) är också det en vidareutveckling av delar från SGML. Standarden utvecklades från början för att klara av storskalig publicering av data (World Wide Web Consortium, 2003a), men används även till publicering av mindre mängder av data. Till skillnad från HTML tillför inte XML någon semantisk information för att tolka struktur och formatering. Detta är en fördel med XML jämfört med HTML i vissa fall eftersom användaren inte är begränsad till ett visst antal taggar för att beskriva struktur och formatering. Ett exempel då XML är mer lämpat än HTML (enligt Elliotte, Rusty, & Harold, 1999) följer nedan. Exemplet visar två möjliga sätt att lagra information om en musiklåt i HTML och XML.

Möjligt sätt att lagra information om en musiklåt i HTML :

```
<dt>Hot Cop
<dd> by Jacques Morali, Henri Belolo, and Victor Willis
<ul>
<li>Producer: Jacques Morali
<li>Publisher: PolyGram Records
<li>Length: 6:20
<li>Written: 1978
<li>Artist: Village People
</ul>
```

Möjligt sätt att lagra information om en musiklåt i XML :

```
<SONG>
<TITLE>Hot Cop</TITLE>
<COMPOSER>Jacques Morali</COMPOSER>
<COMPOSER>Henri Belolo</COMPOSER>
<COMPOSER>Victor Willis</COMPOSER>
<PRODUCER>Jacques Morali</PRODUCER>
<PUBLISHER>PolyGram Records</PUBLISHER>
<LENGTH>6:20</LENGTH>
<YEAR>1978</YEAR>
<ARTIST>Village People</ARTIST>
</SONG>
```

2.1.2.4 XHTML

XHTML (Extensible HyperText Markup Language) är en omformulering och vidareutveckling av HTML baserat på XML. XHTML är tänkt att efterträda HTML (World Wide Web Consortium, 2003b).

2.2 Komprimeringsalgoritmer

Komprimering på datorer innebär att ändra representationen av en fil. Detta resulterar i att det krävs mindre utrymme att lagra och att det tar mindre tid att skicka filen. För att komprimeringen ska var förlustfri måste det också gå att återskapa datan exakt ifrån den komprimerade representationen av den (Bell, Moffat & Witten, 1995). Inom komprimering av text i eBooks är det endast förlustfri komprimering som är intressant, eftersom förlustfull komprimering skulle innebära att texten blev förändrad och läsaren av boken troligtvis inte skulle kunna läsa den.

Komprimeringseffektivitet beräknas genom att jämföra den okomprimerade datan och den komprimerade datan, detta kan göras på några olika sätt. Ett sätt är att beräkna ”bits per char”, det vill säga hur många bitar som den komprimerade datan i genomsnitt använder för att komprimera en byte av den okomprimerade datan. Ett annat sätt är ”återstående procent”, det vill säga hur många procent som återstår i den komprimerade datan jämfört med den okomprimerade datan. Detta beräknas genom att storleken på den komprimerade filen divideras med storleken på den okomprimerade filen (Bell, Moffat & Witten, 1995). Jämförelser i detta arbete kommer att göras i ”återstående procent”.

Inom komprimering av text finns det många kända algoritmer. De vanligaste av dessa, vilka kan vara intressanta för användning på eBooks, kommer tas upp under fortsatta delen av detta kapitel.

2.2.1 Huffmankomprimering

Huffmankomprimering bygger på principen att byta ut representationen för vanligt förekommande data till korta representationer och representationen för ovanligt förekommande data till längre representationer. Komprimeringsmetoden kräver alltså sannolikhetsvärden för den datan som ska komprimeras. Sannolikhetsvärdena som används kan tas fram på olika sätt, det enklaste är att ha en fördefinierad modell. Fördelarna med detta är förutom att det är en lätt metod att använda att det går snabbt att utföra kodning för den givna datan. En ytterliggare fördel är att det bara behöver lagras en modell för avkodning. Nackdelarna är att kodningen oftast inte är optimal eftersom en och samma modell används för all data som ska komprimeras.

En annan metod är att analysera den data som ska komprimeras och bygga upp en modell som är optimal för datan. Fördelarna med detta är att själva komprimeringen blir mer effektiv. Nackdelarna är att modellen måste lagras, vilket tar extra utrymme. Metoden tar också längre tid att utföra än den först nämnda.

En tredje metod är dynamisk Huffmankomprimering, vilket går ut på att en standardmodell används från början som sedan dynamiskt förändras medan komprimering/dekomprimering pågår. Fördelen med detta är att modellen inte behöver lagras och lagringsutrymme sparas. Utförandet av kodningen utförs på liknande sätt som när en tabell med sannolikheter för förekomster av olika data finns. Dynamisk Huffmankomprimering skiljer sig något från de två

andra kodningssätten eftersom uppdateringar av sannolikhetstabellen görs dynamiskt under komprimering och dekomprimering.

2.2.2 Lempel-Ziv komprimering

Eftersom det tar utrymme att lagra tabeller vid komprimering har det kommit fram en metod som kallas adaptivt uppslagsverk (eng. adaptive dictionary). I stort sett alla komprimeringsmetoder som använder sig av detta baseras på metoder som utvecklats av Lempel och Ziv under 1970-talet (Bell, Moffat & Witten, 1995). Metoderna brukar kallas för LZ77 (Lempel & Ziv, 1977) och LZ78 (Lempel & Ziv, 1978) efter de årtal då metoderna publicerades. Båda metoderna använder liknande tillvägagångssätt för att åstadkomma komprimering. En textsträng i den text som ska komprimeras byts ut mot en pekare på det ställe där textsträngen senast förekom i texten. Den tidigare texten används alltså som en slags tabell med textsträngar som kan användas vid komprimering av den resterande texten.

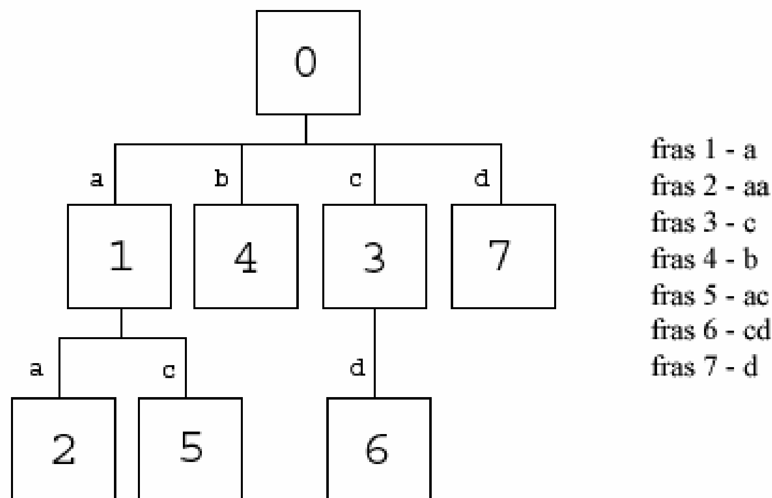
2.2.2.1 LZ77

I LZ77 består pekarna av tre värden. Det första värdet är hur långt tillbaka i texten som textsträngen som ska representeras finns, det andra värdet är längden av textsträngen och det tredje värdet är teckenvärdet som följer efter det som refererats. Det tredje elementet behövs bara om tecknet som ska kodas inte förekommer någonstans i den tidigare texten. Det brukar inkluderas i pekarna vid beskrivning av algoritmen för enkelhets skull, men i implementationer av algoritmen används oftast värdet på en bit för att indikera om det är en pekare eller ett tecken som följer i datan. Ett exempel på en pekare, <5, 3, b>, ska alltså dekomprimeras genom att kopiera tre tecken från fem tecken bakåt i den okomprimerade texten följt av att ”b” läggs till. Rekursiva referenser genom pekare är också tillåtna. Ett exempel på detta är <1, 10, a>, vilket skulle resultera i att tecknet som är ett steg bakåt i texten först skulle kopieras. Nästa steg är att kopiera det tecken som följer det första tecknet, vilket är det tecknet som nyss kopierats. Detta förlopp upprepas tills tio stycken likadana tecken kopierats och sedan skrivs också ”a” (Exempel enligt Bell, Moffat & Witten, 1995, sid 50). LZ77 sätter begränsningar för hur långt bak i filen som pekarna kan referera och hur långa stycken som kan refereras. Detta eftersom ju längre bak som det går att referera och ju större textstycken som kan refereras desto mer utrymme krävs för att lagra pekarna. När text ska komprimeras med LZ77 söks tidigare text igenom för att hitta den längsta matchande textsträngen för den textsträng som följer i texten. En optimal variant av LZ77 skulle vara den variant där det val av längd som pekare kan referera och mängd som går att referera med dem skulle resultera i den bästa komprimeringseffektiviteten på datan. En optimal variant varierar därför mellan vad för typ av data som komprimeras. Att beräkna fram en optimal variant är väldigt resurskrävande och detta används inte i befintliga komprimeringsalgoritmer. En populär komprimeringsalgoritm som baseras på LZ77 är GZip från Gnu Free Software.

2.2.2.2 LZ78

LZ78 använder sig av pekare till ett begränsat antal textsträngar, till skillnad från LZ77 som kan referera ett godtyckligt antal textstycken tidigare i texten. LZ78 har också till skillnad från LZ77 inga begränsningar på hur långt tillbaka i texten som pekare kan referera. LZ78 delar in textstycken i delsträngar, s.k. fraser vilket är det enda som kan refereras av pekarna. Pekarna består därför av endast två värden, det första värdet är vilken fras som refereras och det andra värdet är vilket tecken som följer det som refererats. Ett effektivt sätt att hantera fraserna är

genom att använda en trädstruktur. Ett exempel på ett frasträd kan ses i figur 2. När komprimering utförs gås trädet igenom med hjälp av teckensträngen som ska komprimeras antingen tills en lövnod i trädet påträffas eller tills det inte finns en nod för nästa tecken i strängen. När detta inträffar skrivs frasnummret från den nod i trädstrukturen som genomgången slutade på. Efter detta läses nästa tecken och en ny frasnod lägg till i trädet. Den nya frasen blir frasen som precis skrivits följt utav det nya tecknet. Fras 0 i exemplet är den tomma frasen som inte innehåller något, den används som utgångspunkt i trädet. En populär variant av LZ78 är LZW (Lempel-Ziv-Welch) (Welch, 1984), vilket används av Unix-verktyget compress.



Figur 2 : Frasträd under LZ78 komprimering

2.2.3 Aritmetisk komprimering

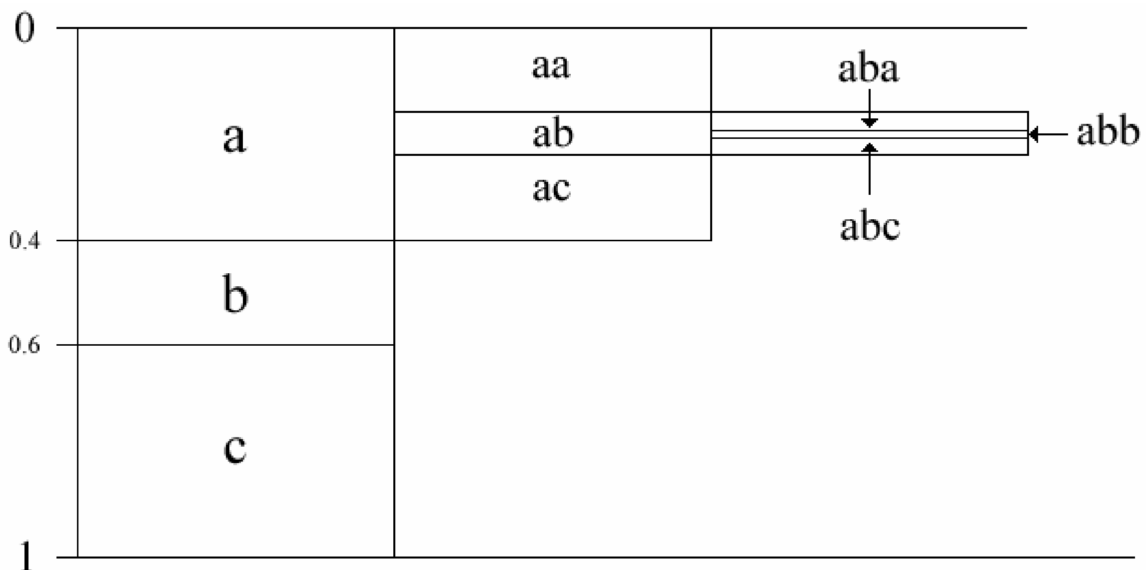
Aritmetisk komprimering använder sig precis som Huffmankomprimering av sannolikhetsvärden för datan som ska komprimeras. Aritmetisk komprimering använder sig av ett intervall av tal mellan 0 och 1 för att representera datan. Desto större datan som ska komprimeras är, desto mindre blir intervallet vilket innebär att fler bitar krävs för att lagra datan. Här följer ett exempel på hur kodning av flera värden går till är att koda ”a” följt av ”b” och ”c”, givet sannolikhetstabellen i tabell 2.

Tecken	Sannolikhet (procent)	Kodnings-intervall
a	40	0.00-0.40
b	20	0.40-0.60
c	40	0.60-1.00

Tabell 2 : Sannolikheter för ett antal tecken

Första utgångspunkt är från intervallet 0 till 1, sedan för att först koda tecknet ”a” begränsas intervallet till 0 till 0,40. Kodningen av ”a” följt av ”b” beräknas genom att intervallet 0 till 0,40 används som utgångspunkt för kodningen när kodningsintervallet för ”b” beräknas in. Kodningen för ”ab” blir då intervallet 0,16 till 0,24. Startvärde för intervallet blir

0+(0,40*0,40), vilket är startvärdet för intervallet för kodningen av ”a” adderat med intervallstorleken för kodning av ”a” multiplicerat med startvärdet för ”b”. Slutvärdet blir 0+(0,40*0,60), vilket är startvärdet för intervallet för kodning av ”a” adderat med intervallstorleken för kodningen av ”a” multiplicerat med slutvärdet för ”b”. Kodningen för ”abc” blir intervallet 0,208 till 0,24. Där startvärdet för intervallet beräknas 0,16+(0,08*0,60) och slutvärdet beräknas 0,16+(0,08*1,00). Ett illustration av hur detta intervall kortas av visas också i figur 3.



Figur 3 : Intervall vid aritmetisk komprimering

2.2.4 Prediction by Partial Match (PPM)

PPM-modellen (Bell & Witten, 1984) upprätthåller statistik för den text som tidigare komprimerats. Men till skillnad från många andra komprimeringsmetoder begränsar PPM sig inte till ett fixt antal tecken för att ta fram sannolikheten för tecknet som ska kodas. Metoden börjar med det största antalet tidigare tecken som statistik upprätthålls för. I följande exempelvis upprätthåller PPM-algoritmen statistik för upp till fyra tecken i följd. Det gäller också att bokstaven ”k” ska komprimeras och de fyra senaste bokstäverna i datan har varit ”stor”. Tidigare i texten har orden ”stork” och ”stora” har förekommit en gång var. Givet detta kan sannolikheten för ”k” sparas som 50 procent. Om det inte finns något sannolikhetsvärde lagrat tidigare för ”k” när de fyra senaste tecknen varit ”stor” minskar metoden antalet tidigare tecken vars statistik jämförs med och algoritmen jämför istället med sannolikheter för tecken som följer textsträngen ”tor”. Minskning av antalet tecken som tas hänsyn till representeras med hjälp av ett specialtecken i den komprimerade datan. Finns ingen statistik för ”k” givet att de tre tidigare bokstäverna var ”tor” skrivs ytterliggare ett specialtecken av samma sort och antalet bokstäver som metoden tar hänsyn till minskas igen, nu ner till 2 tecken, detta upprepas ner tills statistik där inga tidigare tecken tas hänsyn till. Om det inte heller finns någon statistik för tecknet när hänsyn inte tas till någon tidigare text byts sannolikhetsvärden ut enligt en enkel modell där alla tecken har samma sannolikhet. När ett tecken eller en teckensträng har kodats uppdateras sannolikheterna för det respektive tecknet eller teckensträngen. Hur sannolikheten för specialtecknet ska beräknas finns det flera olika metoder för, men dessa gås inte igenom här. Värt att notera är att det är sannolikheten för specialtecknet som skrivs och inte tecknet i sig, detta eftersom PPM ofta kodas med hjälp av

aritmetisk komprimering. Aritmetisk komprimering lämpar sig bra för PPM eftersom givet ett antal tidigare tecken i långa texter är sannolikheten för nästa tecken oftast stor, vilket aritmetisk komprimering hanterar effektivt.

2.3 Komprimeringsverktyg

Under senare år har flera specifika komprimeringsverktyg för XML utvecklats. eBooks använder sig som tidigare nämnts av XHTML vilket i sig är XML. Två av dessa verktyg gås här kort igenom.

2.3.1 XMill

XMill (Liefke, 2000) använder sig av tre olika principer för att förbättra komprimering av data jämfört med traditionella komprimeringsverktyg. Det första som XMill gör är att separera struktur och data, där strukturen består av taggar och attribut till taggarna medan data består av innehållet som taggarna beskriver. Datan och strukturen komprimeras var för sig. Den andra principen som används är att gruppera data som antas vara av likartad typ, som standardinställning i XMill beror detta på vilka sorts taggar som omger datan. Det går att ange vilka taggar som ska komprimeras tillsammans. Om det finns något sådant samband för eBooks skulle komprimeringseffektiviteten kunna förbättras. Komprimering av de olika grupperna görs var för sig. Den sista principen som används är att använda olika semantiska komprimeringsalgoritmer på olika sorters data, heltal för sig, text för sig, datum för sig, och så vidare.

Semantiska komprimeringsalgoritmer utför effektivare omskrivningar av data, vilket görs innan komprimering med traditionella komprimeringsalgoritmer görs. XMill använder sig av GZip vilket bygger på LZ77 (se kapitel 2.2.2.1). Som standard ignorerar XMill mellanslag och använder sig av standardformatering vid dekomprimering av datan, alltså en sorts förlustfull komprimering. Det går att ange vid användning av XMill, vilket behövs för förlustfri komprimering av eBooks, att mellanslag hanteras som vanlig text. XMill stödjer att användare skriver egna semantiska komprimeringsalgoritmer, men detta är bara användbart om den XML-data som ska komprimeras har egenskaper som skiljer sig från vanlig data. Som exempel för användningsområde för nya semantiska komprimeringsalgoritmer ger Liefke (2000) lagring av mätvärden för en termometer där mätvärdena är höga men inte skiljer sig speciellt mycket från andra mätvärden av samma typ. XMill innehåller redan som standard några enkla semantiska algoritmer såsom komprimering av positiva heltal (eng. unsigned int) och uppslagsverkskodning av räknetyper (eng. enumeration values).

2.3.2 XMLPPM

XMLPPM (Cheney, 2001) är ett komprimeringsverktyg för XML som använder sig av metoden PPM (Se kapitel 2.2.4) och ett tillvägagångssätt för att modellera trädstrukturer, MHM (Multiplexed Hierarchical Modeling) vilket Cheney själv utvecklat. XMLPPM använder en liknande idé som XMill, där XML-datan delas upp och olika metoder används för de olika typerna, en modell för element, en modell för struktur och så vidare. Olika komprimeringsalgoritmer körs sedan på de olika typerna av data. XMLPPM använder sig också av de hierarkiska sambanden i XML för att åstadkomma bättre komprimeringsresultat.

3 Problemdefinition

Innehållet i eBooks bör komprimeras för att bättre utnyttja det begränsade lagringsmediet och minnet på vilket de elektroniska böckerna lagras. Lagringsmöjligheterna på dagens befintliga externa eBook-läsare är begränsade. Exempelvis så har Gemstar eBook 1150, Frankling eBookMan 901 och Sony Clié PEG S-320 samtliga 8 mb lagringsutrymme i standardutförande.

Komprimering av eBooks är också intressant för att minska överföringskostnaden när de elektroniska böckerna ska överföras till de externa eBook-läsarna. Detta kan redan i dagsläget göras direkt från Internet med modem eller nätverkskort som finns inbyggt i de externa eBook-läsarna.

Det finns alltså ett behov av att utföra en utvärdering av vilka olika komprimeringsalgoritmer som lämpar sig vid lagring av eBooks.

3.1 Problemspecificering

Målet med arbetet är att undersöka om komprimering av eBooks enligt OeBF kan förbättras genom att ta hänsyn till egenskaper som är specifika för eBooks.

För att nå detta mål delas det upp i tre delmål :

(i) Utföra en litteraturstudie av komprimeringsalgoritmer som kan lämpa sig för användning på den text eBooks enligt OeBF-standarden innehåller. Anledningen till att bild-, video- och ljudkomprimering inte berörs i detta arbete är att de format av dessa tre typer som stöds av OeBF-standarden redan är komprimerade med specialiserade komprimeringsalgoritmer.

(ii) Ta fram en anpassad komprimeringsalgoritm för texten i eBooks.

(iii) Testa egenskaper för den framtagna komprimeringsalgoritmen och jämföra den mot andra existerande komprimeringsalgoritmer. De egenskaper som kommer att tas upp i testet är framförallt komprimeringseffektivitet.

3.2 Förväntat resultat

Resultatet av arbetet kommer att bli en utvärdering av om den anpassade komprimeringsalgoritmen presterar bättre än befintliga algoritmer med avseende på komprimeringseffektivitet. Rapporten kommer också att innehålla en redovisning av resultat av tester utförda på eBooks med några av de undersökta komprimeringsalgoritmerna.

4 Metod

De metoder som kommer användas är en litteraturstudie av komprimering, implementering av komprimeringsalgoritmer och en utvärdering av testningsresultat av olika komprimeringsalgoritmer på eBooks. Litteraturstudien utförs för att få grundläggande kunskaper om komprimering. Implementeringen utförs för att kunna utvärdera komprimeringsegenskaperna för komprimeringsalgoritmen och jämföra dessa mot andra befintliga verktyg.

4.1 Analys av OeBPS

För att utveckla en specifik komprimeringsalgoritm för eBooks undersöks OeBPS v1.2 för att hitta egenskaper som kan utnyttjas för bättre komprimeringseffektivitet. En eBook består av ett antal obligatoriska filer. En av dessa grundläggande filer är den publikationsstruktur innehållande information om samtliga andra filer i boken. Denna fil använder filsuffixet ”opf”. Filen är uppbyggd med XML och har också ett antal standardtaggar som det finns givna tolkning av i OeBPS. eBooks består också till stor del av textfiler som enligt OeBPS ska lagras som XHTML. Andra format är också tillåtna, men då måste också en XHTML-version av samma text finnas. För representation av tecken använder sig OeBPS av Unicode-standarderna UTF-8 och UTF-16. UTF-8 använder sig av en till sex bytes för att koda representationen av Unicode-värden (eng. Unicode scalar value) och UTF-16 använder sig av sekvenser av två bytes för att koda representationer för Unicode-värden. Tabell 3 visar hur de båda standarderna kodar olika teckenvärden, där ”wwww” för UTF-16 på sista raden är ”uuuuu”-1 vilket beror på definitionen av Unicode surrogatpar. Detta är inte relevant för undersökningen eftersom Unicode-värden inte ska tolkas och därför gås inte detta igenom vidare. Då konvertering från UTF-16 till UTF-8 och UTF-8 till UTF-16 lätt kan göras kommer endast hantering av en av dessa två standarder göras vid implementering av algoritmerna. UTF-8-standarden väljs att användas vid fortsatt arbete därför att ASCII-värden har identisk representation i UTF-8. ASCII (American Standard Code for Information Interchange) är en standard för hur 128 tecken ska kodas, bl.a. bokstäver från a till z och siffror. Standarden använder sig av 7 bitar för att representera ett tecken men tecknen lagras vanligtvis som en byte där de värden som används är 0 till 127. Några av tecknen i standarden är kontrollvärden och inga egentliga tecken. Ytterliggare en anledning till att UTF-8 väljs är att de flesta eBooks innehåller framförallt ASCII-värden och lagringen av dessa sker med en byte i UTF-8, medan de tar två bytes i UTF-16.

I eBooks generellt är opf-filen så liten i jämförelse med textfilerna så vid komprimering skulle det ge mycket lite att komprimera den. Innehållet i filen skiljer sig också mycket från vanlig text då den innehåller enbart taggar. Därför kommer inte försök göras för att få denna fil mindre.

Unicode värde	UTF-16	UTF-8 Byte1	UTF-8 Byte2	UTF-8 Byte3	UTF-8 Byte4
00000000 0xxxxxxx	00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
uuuuu zzzzyyyy yyxxxxxx	100110ww wwzzzzyy + 110111yy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

Tabell 3 : UTF-8 och UTF-16 encoding (enligt Unicode Consortium, 2000, sid 37)

4.2 Analys av eBooks enligt OeBPS

eBooks skiljer sig ifrån generell XHTML eftersom användandet av taggar oftast är väldigt lågt. Böckerna innehåller främst långa textstycken. Det verkar därför viktigt att ta hänsyn till vanligt förekommande ord i textstyckena. Runsten (1998) har tidigare undersökt hur kombinationen av Huffmankomprimering följt av Lempel-Ziv komprimering påverkar HTML-dokument innehållande godtycklig text. I sin jämförelse använder sig Runsten av en Huffman-variant som resulterar i att datan som ska komprimeras generellt blir större än den är okomprimerad. När Lempel-Ziv-komprimeringen sedan utförs på data blir den data som komprimerats med Huffman-varianten följt av Lempel-Ziv-komprimeringen större än om enbart Lempel-Ziv-komprimeringen utförs. En undersökning om samma gäller för Huffmankomprimering följt av Lempel-Ziv-komprimering, där Huffmankomprimeringen gör den okomprimerade datan mindre, återstår att utföras. Då det med Huffmankomprimering går att ta hänsyn till att data är vanlig text skulle detta följt av en Lempel-Ziv-komprimering för generell data kunna komplettera varandra väl. Då en optimal Lempel-Ziv-komprimering är väldigt tidskrävande skulle en Huffmanvariant som tar hänsyn till ord i texten också kunna snabba upp komprimeringen.

4.3 Val av algoritmer för implementation

Implementering av två olika komprimeringsalgoritmer för eBooks har utförts. Båda tar hänsyn till förekomsten av ord i texten. Den första algoritmen kodar förekomsten av de mest förekommande orden utan att ta hänsyn till hur vanligt förekommande de är i jämförelse med varandra. Kodning av dessa ord sker med lika många bitar utan hänsyn till vad som är optimalt. Den andra algoritmen tar hänsyn till hur vanligt förekommande de olika orden är och bygger upp ett Huffmanträd för att bestämma hur många bitar som representationen av varje ord ska ha. Effektiviteten hos dessa algoritmer har jämförts, och resultatet av detta gås igenom i kapitel 6. En jämförelse har också gjorts med avseende på hur den komprimerade datan från dessa två komprimeringsalgoritmer påverkas om den följs av en Lempel-Ziv-komprimeringsalgoritm.

5 Genomförande

I detta kapitel tas detaljer upp angående implementeringen av komprimeringsalgoritmerna som valts att utföras. Val som gjorts motiveras och ytterliggare teori som används vid implementering gås igenom.

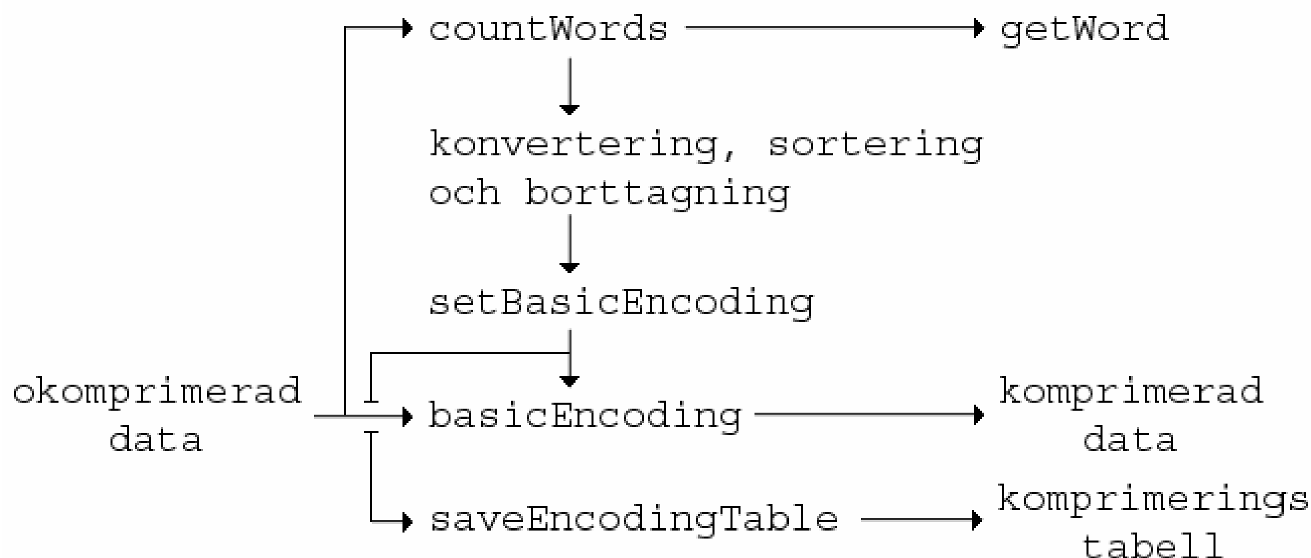
5.1 Implementering av primitiv komprimeringsalgoritm

I detta avsnitt tas detaljer angående den primitiva komprimeringsalgoritmen igenom. Först presenteras hur komprimeringsförloppet går till och sedan hur dekomprimering går till.

5.1.2 Komprimering

Den primitiva komprimeringsalgoritmen som implementeras stödjer inte att Unicode-värden används. Här följer ett förslag till ändring som behövs göras för att filer innehållande Unicode-värden också ska kunna komprimeras med denna algoritm. Unicode Consortium har utvecklat en metod för att komprimera Unicode-texter (Unicode Consortium, 2002). I denna metod använder de sig av ASCII-värden mellan 0 och 31, förutom värdena för tabb (9) och de båda värdena för ny rad (10, 13), för att markera specialtecken i sin komprimering eftersom dessa specialtecken inte används i textfiler. Algoritmen använder sig av dynamiskt placerade fönster. Ett fönster är 128 på varandra följande Unicode-tecken. Algoritmen komprimerar Unicode-tecken som följer varandra och är inom samma fönster. Detta skulle kunna användas på eBooks, men algoritmen komprimerar inte de tecken som ingår i ASCII eftersom dessa representeras på samma sätt som innan. Samma användning av specialvärden inom ASCII skulle gå att använda i andra komprimeringsalgoritmer som utvecklas. ASCII-värdena 16 till 31 är de bytes som börjar med dom fyra bitarna ”0001” (”0001 0000” till ”0001 1111”) och används alltså inte för representation i textfiler. Det skulle alltså gå att använda bitsträngen ”0001” för att representera att data som följer är ett Unicode-värde, som inte ingår i ASCII.

Den primitiva komprimeringsalgoritmen börjar med att söka igenom den okomprimerade filen och räknar hur många gånger varje ord förekommer i filen. En vektor skapas sedan som sorteras med avseende på den storlek som alla förekomster av ordet tar i den okomprimerade filen (antal förekomster av ordet * ordets storlek). Ord som bara förekommer en gång tas bort eftersom det skulle ta mer utrymme att lagra komprimeringsvärdet och ordet okomprimerat i tabell-filen än att lagra ordet okomprimerat i utdata-filen. Efter detta sparas de 128 mest förekommande orden. En kodningsrepresentation tas fram för varje ord och sedan går algoritmen igenom den okomprimerade filen och byter ut de 128 mest förekommande orden mot den kodningsrepresentation som tagits fram tidigare. Efter detta sparar den också den tabell som anger vilka ord som har vilken komprimeringsrepresentation i en separat fil. Förloppet finns beskrivet som ett flödesdiagram i figur 4 och de funktioner som används kommer att beskrivas mer detaljerat.



Figur 4 : Komprimeringsförlopp för primitiv komprimeringsalgoritm.

Funktionen ”countWords” går igenom den okomprimerade datan och sparar alla ord i den som strängar i en tabell tillsammans med antalet förekomster av ordet i datamängden. ”countWords” använder sig av funktionen ”getWord” för att få nya ord från indatan. ”getWord” returnerar ord som endast innehåller siffror, stora och små bokstäver och bindestreck. Andra tecken används som ordseparerare och resulterar i att texten delas upp i ord. Efter ”countWords” tas alla ord utom de 128 som tar störst utrymme i indatan bort, givet att det finns 128 ord i indatan. Detta eftersom de senare 128 bitsträngarna (”1000 0000” till ”1111 1111”) kommer användas för att representera orden komprimerade. Funktionen ”setBasicEncoding” tilldelar de valda orden ett unikt representationsvärde mellan 128 och 255 som kommer användas vid komprimeringen. Funktionen ”basicEncoding” utför själva komprimeringen, den läser indatan och när ett av de ord som ska komprimeras förekommer sparar den representationsvärdet för ordet istället för ordet i utdatan. I övriga fall skrivs indata till utdata oförändrad. Efter detta tar funktionen ”saveEncodingTable” och sparar tabellen för okomprimerat ord med dess respektive representationsvärde så att dekomprimeraren kan tolka de komprimerade orden.

5.1.3 Dekomprimering

Dekomprimeraren läser in tabellen som innehåller information om vilken datarepresentation som tillhör vilket ord. Den går sedan igenom den komprimerade filen och byter ut förekomsten av de komprimerade respresentationerna av ord till motsvarande okomprimerade ord. Detta görs genom att läsa in en byte i taget. Om byte-värdet är mindre än eller lika med 127 är det ett okomprimerat ASCII-värde som skrivs oförändrat till utdatan. I annat fall är det ett komprimerat ord och tabellen används för att undersöka vilket komprimerat ord som byte-värdet representerar varmed det motsvarande okomprimerade ordet skrivs till utdatan.

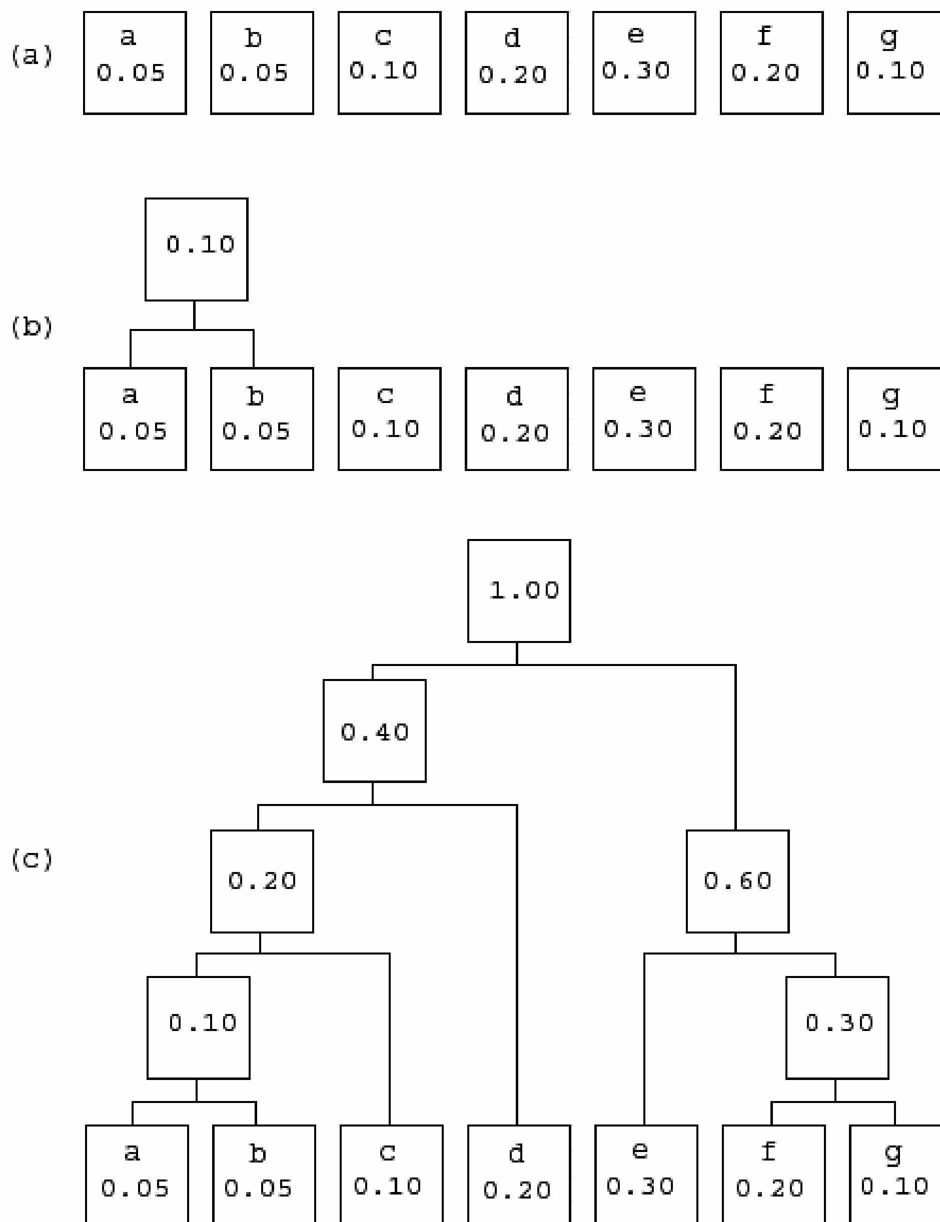
5.2 Implementering av Huffmankomprimeringsalgoritm

I detta avsnitt tas detaljer angående Huffmankomprimeringsalgoritmen igenom. Precis som i kapitel 5.1 presenteras hur komprimeringsförloppet för algoritmen går till och sedan hur dekomprimering går till. Hantering av UNICODE-värden kan hanteras på samma sätt som förslaget som nämnts i 5.1.2, men också i denna algoritm har detta inte implementerats ännu.

5.2.1 Huffmankodning i detalj

För att finna en Huffmankodning så skapas ett Huffman-träd (Bell, Moffat & Witten, 1995). En binär trädstruktur används där sannolikheter för noderna i trädet är lagrade. Sannolikhetsvärdena som används är ett värde mellan 0 och 1, som motsvarar förekomsten av symbolen i texten. Symboler är den data vilken hänsyn tas till när komprimering sker. Symbolerna vid komprimering ska vara ömsesidigt uteslutande, d.v.s. om exempelvis textsträngen ”myra” finns med som komprimeringssymbol i texten ska inte textsträngarna ”my” och ”ra” användas som symboler och vice versa. Om symbolerna inte är ömsesidigt uteslutande och exempelvis ”my” och ”ra” räknas när ordet ”myra” förekommer kommer sannolikhetsvärdena vara missvisande och trädet som genereras blir mindre optimalt eftersom ”my” och ”ra” kan knuffa ner mindre förekommande ord så att de får längre kodningsrepresentation. I algoritmerna tas hänsyn till ord genom att använda mellanrum och andra tecken för att separera ord. Därigenom utesluts att delar av ord räknas när hela ord förekommer.

Vid generering av trädet skapas först lövnoder till trädet. Ett exempel på hur detta kan se ut visas i figur 5a. Nästa steg är att skapa en ny nod i trädet och koppla ihop det som förälder till de två noder med lägst sannolikhetsvärde. Sannolikhetsvärdet för den nya noden är summan av sannolikhetsvärdena för de två noderna som kopplas ihop som barn till den nya noden. Ett exempel på hur detta kan se ut visas i figur 5b. När två noder har kopplats ihop med en förälder tas dessa två noder bort som möjliga noder att para ihop och föregående steg upprepas tills det bara finns en möjlig nod att para ihop, vilken är rotnoden till trädstrukturen. Ett exempel på ett Huffmanträd efter ihopkoppling kan ses i figur 5c. Värt att notera är att anledningen till att sannolikhetsvärdet för rotnoden i exemplet blir 1 är att alla möjliga ömsesidigt uteslutande symboler finns med i trädet. Så kommer oftast inte vara fallet när komprimering utförs med de två algoritmerna som beskrivs i kapitel 5.



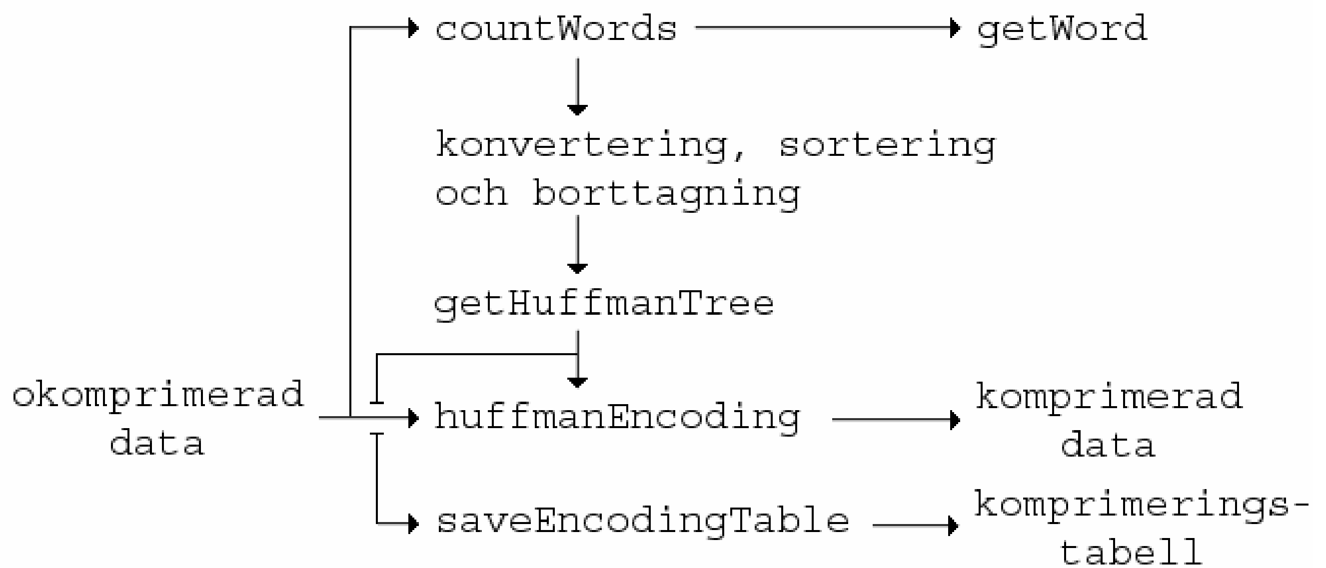
Figur 5 : Huffmanträd (enligt Bell, Moffat & Witten, 1995, sid 32)

För att ta fram bitsträngar för kodning av varje bokstav i exemplet gås trädet igenom från rotnoden till lövnoden för bokstaven. Vid varje nod som gås igenom läggs en nolla till i bitsträngen om ett steg åt vänster tagits för att komma till noden eller en etta om ett steg åt höger tagits för att komma till noden. Utgående från trädet i figur 5c så blir kodningen för ”f” ”110” och kodningen för ”d” blir ”01”.

5.2.2 Komprimering

Huffmanalgoritmen använder samma steg för att ta fram ord ur filen som den primitiva komprimeringsmetoden. Den söker igenom den okomprimererade filen och räknar ord och sorterar dem med avseende på deras storlek, men sedan skiljer sig tillvägagångssättet från den primitiva komprimeringsalgortimen. Huffmanalgoritmen tar inte bort ord så att ett fixt antal

återstår och använder sig inte av någon begränsning för antal tillåtna ord. Tecknet mellanslag läggs också till som ett eget ord eftersom detta tecken är så frekvent förekommande i textfiler att det skulle gå att spara utrymme genom att hitta en kortare representation för det vid komprimering. Ord som bara förekommer en gång tas bort av samma anledning som i den primitiva komprimeringsalgoritmen. Detta följs av generering av ett Huffmanträd, givet sannolikheter för orden i den okomprimerade datan. Generering av Huffmanträdet, till skillnad från exemplet i figur 5, görs med avseende på ord och inte bokstäver. Huffmanträdet används för att ta fram en tabell innehållande ord och respektive komprimeringsrepresentation (en bitsträng av godtycklig längd). Komprimeringen utförs och tabellen sparas separat. Förloppet finns beskrivet som ett flödesdiagram i figur 6 och de funktioner som används beskrivs här mer detaljerat.



Figur 6 : Komprimeringsförlopp Huffmankomprimeringsalgoritm

”countWords” som används är samma funktion som i den primitiva komprimeringsalgoritmen, denna tar fram statistik om den okomprimerade datan och funktionen behöver därför inte anpassas på något sätt till Huffmanalgoritmen. Den konvertering, sortering och borttagning som utförs är också samma som i tidigare algoritm, men antalet ord som lagras har inget maximumvärde till skillnad från den primitiva algoritmen. ”getHuffmanTree” tar emot en tabell med ord och respektive sannolikhetsvärden och genererar ett Huffmanträd enligt algoritmen som nämnts i avsnitt 5.2. Detta träd baseras på storleken av det totala antalet förekomster av ordet i filen.

Ett mer optimalt sätt skulle vara att gå vidare när ett första träd finns och flytta om i trädet så att värdet som trädet baseras på är antal förekomster i filen multiplicerat med storleken av ordet okomprimerat minus komprimeringsrepresentationen (antal förekomster av ordet * (ordets representation okomprimerat – ordets representation komprimerat)). Då skulle de värdena som trädet baseras på att motsvara hur mycket utrymme som sparas på att komprimera olika ord. Problemet med att göra detta från början är att storleken av komprimeringsrepresentationen för orden är okänt tills ett första träd genererats och om trädet byggs om för att få bättre värden kan också detta träd förbättras eftersom det nya trädet

baseras på längden av komprimeringsrepresentationer som orden hade i det tidigare trädet. Efter att ha upprepa detta steg ett antal gånger kommer fortfarande komprimeringsrepresentationerna för orden baseras på gamla träd, men trädet bör ha blivet mer optimalt än utgångsträdet.

För att lätt kunna koda orden görs en tabell för ord och respektive komprimeringsrepresentation som tas fram från Huffmanträdet. För att kunna spara komprimeringsrepresentationen på ett effektivt sätt skapas två klasser som tar hand om att skriva och läsa bitvis från filer. I vanliga fall skrivs bara bytes till filer, men om detta skulle göras i fallet med Huffmankomprimering skulle mycket onödig information lagras vilket skulle försämra komprimeringseffektiviteten. ”OHandler”-klassen tar hand om att skriva datan vid komprimeringen. Klassens skrivningsfunktion tar emot en bool-vektor och skriver till den komprimerade filen när en hel byte har mottagits. Det är inte bara vid skrivning av komprimeringsrepresentationer som klassen måste användas eftersom skrivning av komprimerade ord förskjuter resterande data så det inte går att skriva den okomprimerade datan i den komprimerade filen bytevis. Efter att skrivning av den komprimerade filen slutförts anropas ”saveEncodingTable” och orden som komprimerats skrivs tillsammans med sin komprimeringsrepresentation till tabellfilen.

5.2.3 Dekomprimering

Dekomprimeringsalgortimen börjar med att läsa in komprimeringsrepresentation och motsvarande ord i en tabell. Också dekomprimering av Huffmankomprimeringen kräver speciell hantering eftersom det inte går att läsa in och tolka den komprimerade datan byte för byte. Dekomprimeraren måste också kunna tolka bitsträngar av annan längd än 8 bit (en byte), detta hanteras av klassen ”IHandler”. Komprimerade ord tolkas genom att bit för bit läses in och läggs till i en temporär bitsträng tills bitsträngen matchar ett komprimerat ord eller ett okomprimerat tecken. När den temporära bitsträngen matchar ett komprimerat ord skrivs ordet okomprimerat till utdata-filen och om bitsträngen matchar ett okomprimerat tecken skrivs tecknet oförändrat. Detta följs av att den temporära bitsträngen nollställs och tolkning av datan fortsätter tills filen är slut. Den sista datan i den komprimerade filen går inte alltid tolka då det måste skrivas hela bytes till filer och det inte alltid är hela bytes som komprimeraren behöver skriva. Upp till sju av de sista bitarna i komprimerade filer kan alltså vara oanvända.

6 Resultat

För att kunna dra någon slutsats om huruvida de komprimeringsalgoritmer som utvecklats följt av en Lempel-Ziv-baserad komprimeringsalgoritm presterar bättre än enbart en Lempel-Ziv komprimeringsalgoritm utförs en mätning av komprimeringseffektivitet på ett antal eBooks. Test utförs också med komprimeringsverktyget XMill för att ha ytterliggare värden att jämföra med.

De olika komprimeringsmetoder som testas på de elektroniska böckerna är som följer:

- Den primitiva komprimeringsmetoden
- Den primitiva komprimeringsmetoden följd av GZip (version 1.2.4)
- Den primitiva komprimeringsmetoden följd av Unixverktyget compress (version 4.0)
- Huffmankomprimering
- Huffmankomprimeringen följd av GZip (version 1.2.4)
- Huffmankomprimeringen följd av Unixverktyget compress (version 4.0)
- Enbart GZip (version 1.2.4)
- Enbart Unixverktyget compress (version 4.0)
- XMill (version 0.7b, för detaljer se kapitel 2.3.1)

Test med XML-komprimeringsverktyget XMLPPM uteblev då tillgång till testsystem för test med detta verktyg inte fanns vid utförandet av arbetet.

De eBooks som väljs att utföra testerna på är :

1. 10 kapitel av ”Robinson Crusoe” (Daniel Defoe), 318 341 bytes
2. ”The hitchhiker's guide to the galaxy” (Douglas Adams), 316 637 bytes
3. “2001 : A space odyssey” (Arthur C. Clarke), 408 814 bytes

För tydlighets skull poängteras att det är enbart XML-filerna i böckerna som komprimeras och att böckerna inte innehåller några UNICODE-värden. Då böcker som följer standarden ännu är svåra att tillgå har endast en bok som följer standarden hittats, ”Robinson Crusoe”. De andra böckerna har konverterats till att följa OeBPS-standarden från ren text. En anledning till att inte fler böcker hittats är att möjlighet att köpa böcker som följer standarden inte fanns. Att samtliga tre böcker som testas är skönlitterära anses inte påverka testresultaten eftersom samma resultat förväntas på alla böcker som innehåller många upprepningar av samma ord, eftersom det är det de algoritmer som implementeras tar hänsyn till. De komprimeringsalgoritmer som körs efter de algoritmer som implementerats hanterar generell data så inte heller här förväntas försämring om andra eBooks än skönlitterära komprimeras. De implementerade algoritmerna förväntas prestera sämre på böcker vars innehållande ord inte upprepas ofta.

Tabell 4a och 4b visar filstorleken på filerna efter komprimering och inom parantes visas återstående procent (storleken på den komprimerad datan dividerat med storleken på den okomprimerade datan) avrundat till närmsta heltal. I samtliga tester med den primitiva komprimeringsalgoritmen och Huffmankomprimeringsalgoritmen inkluderar värdena både

tabellfil och datafil. Vid komprimering med compress och GZip har tabell och datafilerna komprimerats separat och filstorleken efter komprimering har summerats.

Bok	Primitiv	Primitiv + GZip	Primitiv + compress	XMill
1	248 332 (78%)	100 926 (32%)	110 481 (35%)	106 864 (34%)
2	266 944 (84%)	109 895 (35%)	117 497 (37%)	115 245 (36%)
3	341 378 (84%)	136 610 (33%)	144 740 (35%)	143 186 (35%)

Tabell 4a : Testresultat för primitiv komprimeringsalgoritm och XMill

Bok	Huffman	Huffman+Gzip	Huffman+compress	Compress	GZip
1	180 257 (57%)	130 766 (41%)	149 214 (47%)	115 883 (36%)	106 866 (34%)
2	201 147 (64%)	143 012 (45%)	161 151 (51%)	120 939 (38%)	113 581 (36%)
3	247 288 (60%)	174 756 (43%)	196 145 (48%)	150 393 (37%)	142 316 (35%)

Tabell 4b : Testresultat för Huffmankomprimeringsalgoritm och separat komprimering med compress och GZip

För att illustrera värdena skapas ett stapeldiagram för varje bok (Figur 7a, 7b och 7c) med samtliga testresultat för respektive bok. I diagrammen används följande förkortningar för komprimeringsalgoritmerna:

P. Primitiv

PG. Primitiv följt av GZip

PC. Primitiv följt av compress

XM. XMill

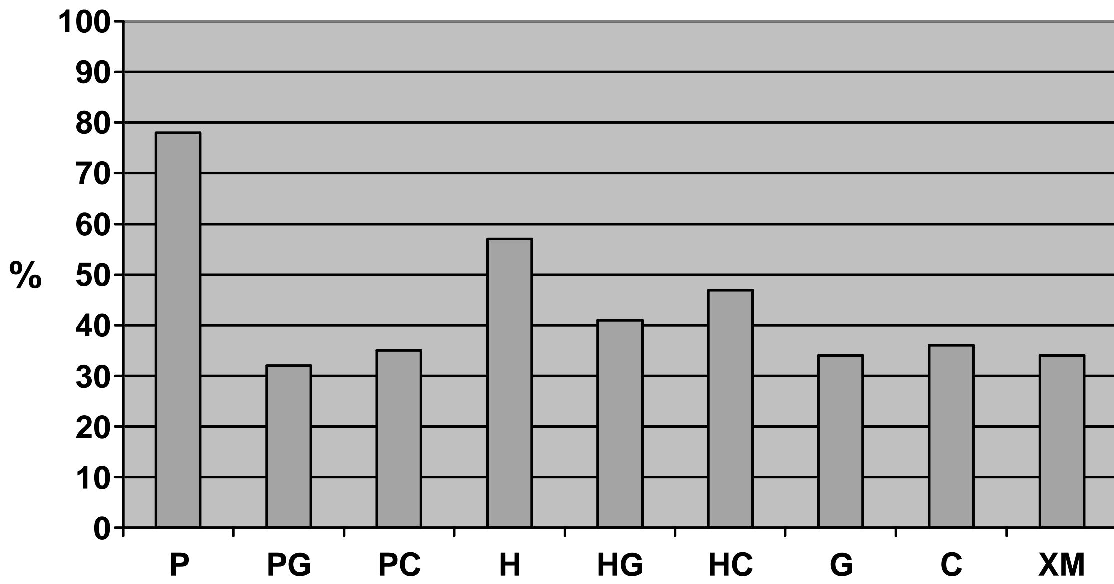
H. Huffman

HG. Huffman följt av GZip

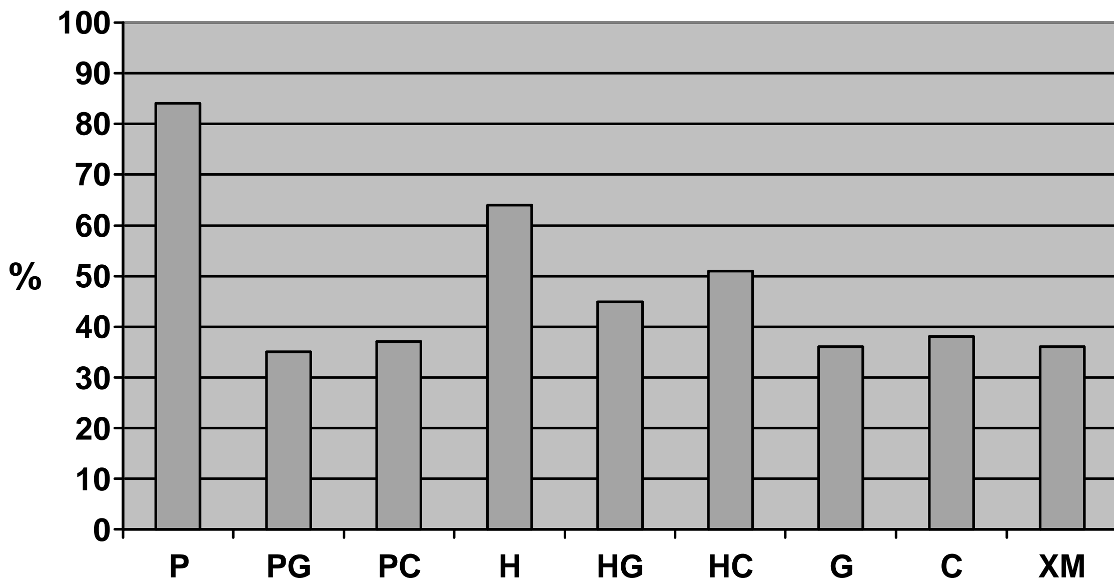
HC. Huffman följt av compress

C. Compress

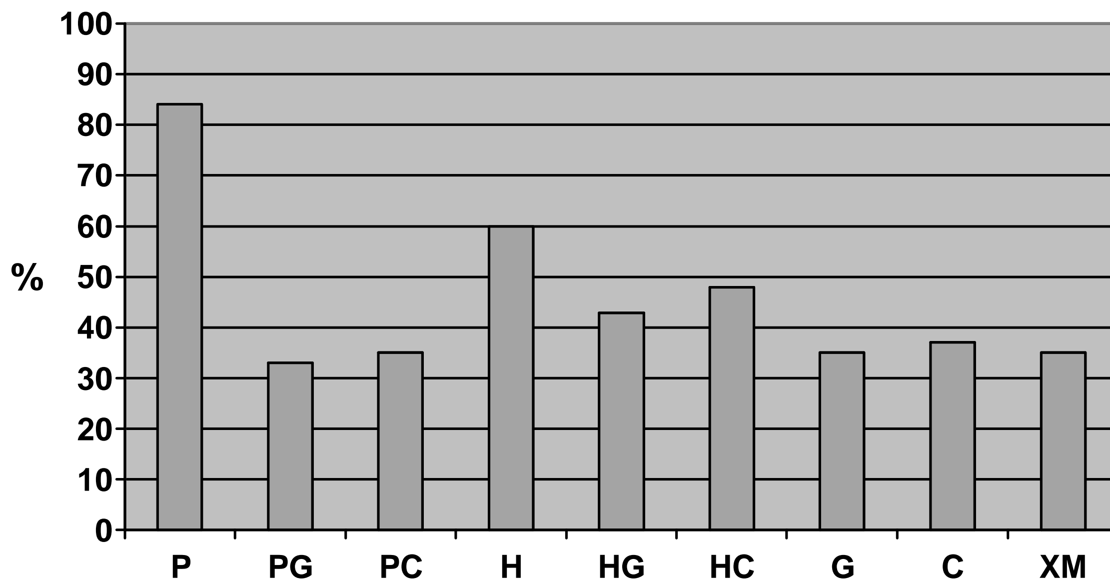
G. GZip



Figur 7a : Testresultat för ”Robinson Crusoe”



Figur 7b : Testresultat för ”Liftarens guide till galaxen”



Figur 7c : Testresultat för ”2001 : A Space Odyssey”

Testresultaten är likvärdiga i testerna på de tre böckerna. Bäst på samtliga böcker presterar den primitiva komprimeringsalgoritmen följt utav GZip. Komprimeringseffektiviteten för den primitiva komprimeringsalgoritmen följt av GZip är några procent bättre än enbart GZip eller XMill, vilka är de komprimeringssätt som presterar näst bäst. XMill använder sig som nämnt i kapitel 2.3.4 av GZip, men på de eBooks tester utförts presterar XMill själv inte någon signifikant förminskning av någon av böckerna. På två av böckerna presterar XMill till och med sämre än enbart GZip. Komprimeringsverktyget compress presterar sämre än GZip i alla kombinationer som prövats. Separat presterar Huffmankomprimeringsalgoritmen bättre än den primitiva komprimeringsalgoritmen, men när komprimeringen följs upp med GZip och compress blir datan komprimerad med Huffmankomprimeringen större än om den primitiva algoritmen används. Huffmankomprimeringen följt av GZip och compress presterar sämre än att enbart använda GZip eller compress i samtliga fall. Den primitiva komprimeringsmetoden förbättrar komprimeringseffektiviteten för både compress och GZip i samtliga testfall.

7 Sammanfattning

För att sammanfatta det arbete som utförts diskuteras de testresultat som finns i tidigare kapitel och förslag till framtida arbete ges.

7.1 Diskussion

Testresultaten blev annorlunda än vad som förväntats. Den primitiva komprimeringsalgoritmen presterade bättre än Huffmankomprimeringen när komprimeringen följdes upp av en Lempel-Ziv-algoritm. Huffmanalgoritmen försämrade komprimeringen för Lempel-Ziv-algoritmerna i samtliga fall. En idé om varför resultaten för Huffmankomprimering inte var lyckat ihop med Lempel-Ziv-algoritmerna är att Lempel-Ziv-algoritmerna arbetar på bytenivå, dvs. att de tolkar en byte i taget. Eftersom den Huffmanalgoritm som utvecklats jobbar på bitnivå förstör den en hel del av de mönster som Lempel-Ziv-algoritmerna utnyttjar. Ett exempel på detta är att ord som komprimeras på bytenivå kodas med samma byteföljd i samtliga fall i den komprimerade filen. Samma ord kan ha kodats på flera olika sätt om datan ses på bytenivå av Huffmanalgoritmen beroende på vad för data som komprimerats innan och vad för data som komprimeras efter ordet. Lempel-Ziv-algoritmerna kan därför inte skriva sina referenser lika ofta eftersom orden oftast inte kan ses som samma ord av Lempel-Ziv-algoritmerna när de är komprimerade på bitnivå. Ett möjligt sätt att förbättra detta vore att utveckla en komprimeringsalgoritm som arbetar enligt Lempel-Ziv-algoritmernas principer men istället söker efter strängar på bitnivå.

En tanke om varför den primitiva algoritmen följt av GZip presterar bättre än enbart GZip separat är, som tagits upp i kapitel 2.2.2.1, att LZ77 algoritmer bara kan referera en begränsad längd bakåt i filen med pekarna. Den primitiva komprimeringsalgoritmen byter ut de mest förekommande orden i hela filen och kan därför öka effektiviteten hos LZ77-algoritmen i vissa fall eftersom orden bara behöver skrivas okomprimerat i tabellfilen en gång medan det är möjligt att det måste skrivas med LZ77-algoritmen mer än en gång. Att den primitiva algoritmen också förbättrar compress skulle kunna bero på att begränsningar har satts på den data som komprimeras, endast de första 128 tecknen i en byte får förekomma och därför kan den primitiva komprimeringsalgoritmen använda en mer effektiv representation av de ord som komprimeras än GZip och compress kan.

7.2 Framtida arbete

Under utveckling och testning av algoritmerna har det uppkommit ett antal idéer om framtida arbete som återstår att undersökas och om hur detaljer i detta arbete skulle kunna förbättras. Beskrivning av dessa idéer följer nedan.

- Optimering av lagring i tabellfil i den primitiva komprimeringsalgoritmen kan göras. Då arbetet som utförts inte fokuserade så mycket på den primitiva komprimeringsalgoritmen utfördes inte denna optimering. Som tabellfilen är i den befintliga implementationen lagras de ord som komprimeras följt av ett mellanslag och det teckenvärde som ordet använder vid komprimering. En optimering av detta skulle vara att skriva enbart de okomprimerade orden separerade med mellanslag. Kodningsrepresentation behöver inte lagras om orden skrivs i följd från ordet med kodningsvärde 128 fram till sista ordet. Detta går alltid eftersom orden som ska komprimeras alltid tilldelas ett komprimeringsvärde stigande från 128.

- Att lägga med stöd för komprimering av taggar. Även om taggar inte är så frekventa i eBooks förekommer de och i vissa fall kan de troligtvis vara frekventa nog att utrymme skulle kunna tjänas på att komprimera dem. Ett enkelt sätt att göra detta skulle vara att acceptera ”<” och ”>” som bokstäver i de befintliga algoritmerna.
- Stöd för alla UNICODE-värden finns inte då det inte ansågs nödvändigt för att kunna utföra tester på befintliga eBooks. Många eBooks använder sig inte av UNICODE-värden, men enligt OeBPS är det tillåtet. Ett förslag på hur detta skulle kunna hanteras finns i avsnitt 5.1.2.
- Att utveckla en variant av LZ77 som tar hänsyn till strängar bitvis istället för bytevis som befintliga implementationer gör. Detta skulle kunna förbättra komprimeringseffektiviteten när metoden att använda Huffmanalgoritmen följt av en Lempel-Ziv-algoritm används. Detta eftersom det finns mönster som Lempel-Ziv-algoritmerna som arbetar på bytenivå inte tar hänsyn till.
- Att utföra test med XMLPPM då detta verktyg inte testat och jämförts med komprimeringsalgoritmerna som implementerats under detta arbete. Test av andra komprimeringsverktyg som inte tagits upp vid testningen i denna rapport skulle också kunna vara intressant för att jämföra med de algoritmer som implementerats.

Referenser

Bell, Cleary & Witten (1990) *Text Compression*. Prentice-Hall Inc.

Bell, Moffat & Witten (1995) *Managing gigabytes : compressing and indexing documents and images*. Van Nostrand Reinhold.

Bell & Witten (1984) Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, COM-32, 396-402. Tillgänglig på Internet: http://cs.haifa.ac.il/courses/src_coding/ClearyWitten1984BasicPPM.pdf [hämtad 03.05.20]

Burnard, L. (1995) *What is SGML and how does it help?* Text Encoding Initiative Consortium. Tillgänglig på Internet : http://www.tei-c.org/Vault/ED/EDW25/index.html [hämtat 03.03.18]

Carvajal, D. (2000, 16 mars) Long line online for Stephen King e-novella. *New York Times*. Tillgänglig på Internet : http://query.nytimes.com/search/article-page.html [hämtad 03.02.26]

Cheney J. (2001) Compressing XML with multiplexed hierarchical models, *Proceedings of the 2001 IEEE Data Compression Conference*, 163-172. Tillgänglig på Internet : http://www.cs.cornell.edu/People/jcheney/papers/cheney-dcc2001.ps [hämtad 03.03.02]

Elliotte, Rusty & Harold (1999), *XML bible*. John Wiley & Sons.

Franklin Electronic Publishers (2003) *Franklin electronical publishers eBookMan 901*. Tillgänglig på Internet: http://www.franklin.com/estore/details.asp?ID=EBM-901 [hämtad 03.02.19]

Gemstar TV Guide International (2003) *Gemstar eBook 1150*. Tillgänglig på Internet: http://www.gemstar-ebook.com/ebcontent/devices/1150_spec.asp [hämtad 03.02.19]

King, S. (2000) *Riding the Bullet*. Scribner.

Lempel & Ziv (1977) A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23, 337-343. Tillgänglig på Internet: http://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf [hämtad 03.05.19]

Lempel & Ziv (1978) Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*, IT-24, 530-536. Tillgänglig på Internet: http://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1978_variable-rate.pdf [hämtad 03.05.19]

Liefke (2000) A Extensible compressor for XML data. ACM SIGMOD Record, 29(1). Tillgänglig på Internet : http://www.cis.upenn.edu/~liefke/papers/sigrec00.pdf [hämtad 03.05.20]

Open eBook Forum (2002) *Open eBook forum publication structure specification* (v1.2). Tillgänglig på Internet: http://www.openebook.org/oebps/oebps1.2/download/oeb12-xhtml.htm [hämtad 03.01.26]

Runsten, M. (1998) *Komprimering av HTML-dokument.* Institutionen för Datavetenskap vid Högskolan i Skövde. Tillgänglig på Internet : http://www.ida.his.se/ida/htbin/exjobb/1998/HS-IDA-EA-98-118 [hämtad 03.04.19]

Sony Electronics Inc. (2003) *Clié PEG S320.* Tillgänglig på Internet: http://www.sel.sony.com/SEL/consumer/ss5/office/clietm/clietmhandheld/peg-s320_specs.shtml [hämtad 03.02.19]

Unicode Consortium (2000) *The Unicode Standard, Version 3.0*. Addison-Wesley Pub Co. Tillgänglig på Internet : http://www.unicode.org/book/u2.html [hämtad 03.04.10]

Unicode Consortium (2002) *Unicode technical standard #6 : A standard compression scheme for Unicode*. Tillgänglig på Internet: http://www.unicode.org/reports/tr6/ [hämtad 03.04.24]

Welch (1984) A Technique for high performance data compression. *IEEE Computer*, 17, 8-20. Tillgänglig på Internet: http://www.cs.duke.edu/courses/spring03/cps296.5/papers/welch_1984_technique_for.pdf [hämtad 03.05.19]

World Wide Web Consortium (2003a) *Extensible Markup Language (XML)*. Tillgänglig på Internet: http://www.w3.org/XML/ [hämtat 03.03.31]

World Wide Web Consortium (2003b) *WC3 HTML Home Page*. Tillgänglig på Internet: http://www.w3.org/MarkUp/ [hämtat 03.04.10]