Linköpings universitet, 2018 | Institutionen för systemteknik

TSBK08 - Datakompression

Entropi-estimering, Huffman-kodning och LZW-kodning

Fredrik Wallström

Handledare : Harald Nautsch
Examinator : Harald Nautsch

LINKÖPINGS UNIVERSITET

Linköpings universitet
SE–581 83 Linköping
013-28 10 00 , www.liu.se

Sammanfattning

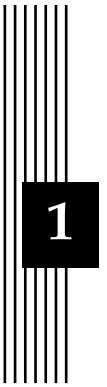
Abstract.tex

Författarens tack

Framför ett tack till Harald Nautsch för en väl genomförd kurs, TSBK08 - Datakompression.

Innehåll

Figurer



1 Introduktion

1.1 Motivation

Att komprimera filer är väldigt användbart i dagens samhälle. Dels för att spara filer i ett mindre format, vilket leder till de tar upp mindre plats på lagringsenheten, och för att det går snabbare att överföra informationen mellan olika användare. Det finns flertalet metoder för att komprimera filer och olika metoder passar bättre till vissa filtyper. Den här rapporten kommer att undersöka två olika komprimeringsmetoder, Huffman-kodning och Lempel-Ziv-Welch-kodning (LZW).

1.2 Mål

Målet med denna rapport är att undersöka två olika komprimeringsmetoder, Huffman-kodning och LZW-kodning. Hur fungerar dessa komprimeringsmetoder och vilka olika fördelar och nackdelar finns, samt hur bra de presterar vid komprimering av olika filtyper.

1.3 Frågeställningar

De frågeställningar som denna rapport kommer att behandla listas nedan.

1. Hur stor plats på lagringsenheten sparar vi då en fil komprimeras med Huffman-kodning respektive LZW-kodning.
2. Hur nära kommer vi den optimala takten (entropin) när vi komprimerar en fil med Huffman-kodning respektive LZW-kodning.
3. Vilka filtyper passar bäst att komprimera med Huffman-kodning respektive LZW-kodning.

1.4 Avgränsningar

Det finns flera olika metoder för komprimering av filer. Denna rapport undersöker endast Huffman-kodning och LZW-kodning och tar ej hänsyns till övriga existerande metoder för

komprimering. Tilläggas bör att vid referering till Huffman-kodning menas endast statisk Huffman-kodning, det vill säga då källans frekvenser är kända på förhand och uppskattas inte samtidigt som filen kodas.

2 Teori

Nedan presenteras hur Entropi-estimering, Huffman-kodning och LZW-kodning fungerar.

Entropi-estimering

Det går att mäta hur bra komprimerad en sekvens är, detta gör man genom mäta takten på koden. Takten på koden ges av [2].

$$Takt\ R = \frac{genomsnittligt\ antal\ bitar\ per\ kodord}{genomsnittligt\ antal\ symboler\ per\ kodord} \quad Bitar\ per\ symbol$$

Det ger alltså ett tal som säger i snitt hur många bitar vi har kodat varje symbol med.

Entropin för en källa är en teoretisk lägre gräns för hur många bitar vi behöver för att koda varje symbol. Takten kan alltså inte vara under denna gräns. Entropin av en källa ges av [2].

$$Entropy\ H(X) = -\sum p(X) \log p(X)$$

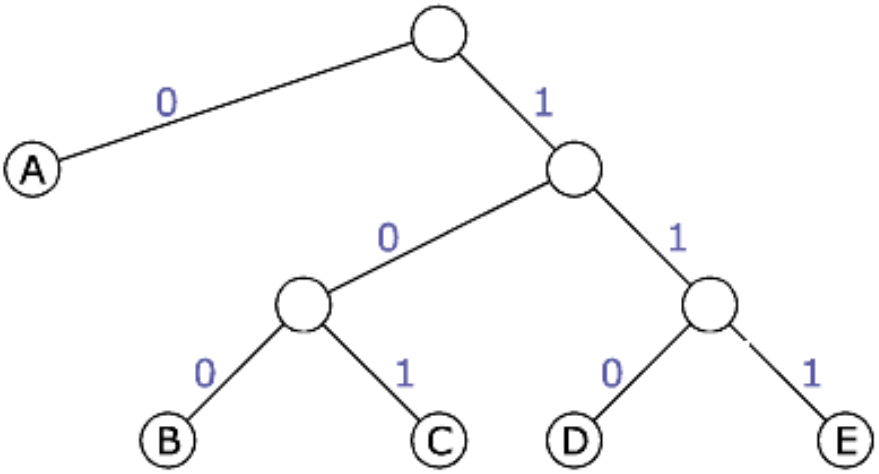
Där p(X) är sannolikheten för given symbol och log är bas 2 logaritm. Vi summerar över alla symboler i källan.

Huffman-kodning

Huffman-kodning är en metod för att komprimera filer till ett mindre format. Metoden utvecklades av den dåvarande studenten David A. Huffman år 1952. Idén med metoden är att tilldela varje symbol som ska kodas en sekvens av olika bitar, 0 eller 1, exempel:

Symbol A kodas till 10101010
Symbol B kodas till 1000000
Symbol C kodas till 11111111

I metoden kommer de symboler som förekommer flest gånger i källan, det vill säga den symbol som har högst sannolikhet, att tilldelas minst antal bitar. Det betyder att de symboler



Figur 2.1: Ett exempel på ett Huffman-träd.

som förekommer minst gånger i källan kommer tilldelas mest antal bitar. Exempel kan symbolen D kodas med bitarna 10 och symbolen E kodas med bitarna 1001010101. Det betyder att symbolen D förekommer oftare i sekvensen än symbolen E. Alltså kommer sekvensen att komprimeras på bästa möjliga vis. Redan här inses att med Huffman-kodning behövs information om hur källan ser ut. Det vill säga, man behöver veta sannolikheterna för de olika symbolerna i källan. Nedan beskrivs mer ingående hur komprimering och avkomprimering fungerar i Huffman-kodning [1].

Komprimering

Att komprimera en källa med Huffman-kodning förutsätter att det finns en given sannolikhetstabell för källan. Det finns alltså en lista som mappar en symbol mot dess sannolikhet, vi kallar här varje symbol och dess sannolikhet för en nod. Metoden fungerar då enligt följande:

1. Ta ut de två symbolerna (noderna) som har lägst sannolikhet i listan.
2. Addera dessa noders sannolikheter och lägg till den som en ny nod i listan, denna nod får ingen symbol. Resterande noder behålls.
3. Börja sedan om från ett om det finns fler än en stycken noder kvar i listan.

Med den här metoden byggs en trädstruktur upp över hur de olika symbolerna kan tilldelas olika bitar. Lövnoderna i trädet representerar de olika symbolerna. Roten av trädet representeras av en nod med sannolikhet 1 eftersom summan av alla sannolikheter i en sekvens alltid summeras till 1. För att tilldela en sekvens av bitar till alla symboler traverserar man igenom hela trädet och bildar en lista som mappar respektive symbol sin egna unika sekvens av bitar. Här är det förutbestämt att varje gång man går vänster i trädet adderar man en 0:a till sekvensen av bitar och varje gång man går höger adderar man en 1:a till sekvensen, eller vice versa. Figur 2.1 visar hur ett Huffman-träd kan se ut [1].

När mappningen mellan symbol och sekvens av bitar existerar kan den givna källan komprimeras. Detta görs genom att läsa av en symbol i i taget ifrån källan och sedan matcha den symbolen med symbolerna som finns i listan som mappar symboler mot en sekvens av bitar. Då ges en sekvens av bitar som adderas till den resulterande bitsekvensen. Denna procedur upprepas tills det inte finns fler symboler att läsa av i sekvensen som ska komprimeras.

Avkomprimering

För att avkomprimera en Huffman-kod behövs samma sannolikhetslista som användes vid komprimeringen av en källa. Förutsatt att denna lista är känd, kan ett Huffman-träd byggas upp på samma vis som vid komprimeringen. Med Huffman-trädet kan man sedan traversera ifrån roten och ner beroende på vilken bit, 0 eller 1, man stöter på i den sekvens som ska avkodas. När man sedan kommer till en lövnod betyder det att en symbol kan avkodas, denna symbol adderas sedan till den resulterande sekvensen. Proceduren startar sedan om ifrån rotnoden och avslutas då det inte längre finns bitar kvar att läsa av i bitsekvensen [1].

LZW-kodning

LZW-kodning är en metod för att komprimera filer till ett mindre format. Metoden utvecklades av Abraham Lempel, Jacob Ziv, och Terry Welch år 1984. Metoden är en utveckling av den tidigare metoden LZ78 metoden som utvecklades av Abraham Lempel och Jacob Ziv år 1978. Idén med LZW komprimering är att skapa en ordbok för olika symboler och kombinationer av symboler. Med kombinationer av symboler menas symboler som förekommer efter varandra i källan och inte alla möjliga kombinationer av symboler ifrån källan. Nedan beskrivs mer ingående hur komprimering och avkomprimering fungerar i LZW-kodning [3].

Komprimering

För att komprimera en given källa med LZW följs dessa steg.

1. Initialisera ordboken så den innehåller alla symboler av längd 1, lämligtvis alla möjliga bytes, 0-255
2. Hitta den längsta möjliga strängen, W, av symboler i ordboken som matchar den aktuella strängen som avläses.
3. Addera motsvarande index i ordboken för W till resultatet.
4. Addera W plus nästa symbol ifrån källan som avläses till ordboken.
5. Gå till steg två så länge som det finns kvar symboler att läsa i källan.

På detta vis erhålls en sträng med index som för sig motsvarar en eller flera symboler [3].

Avkomprimering

Avkomprimeringen för en sekvens med LZW-kodning fungerar som följer. Först intialiseras samma ordbok som användes vid komprimeringen, med undantag att man nu mappar index mot en sträng istället för tvärtom. Man läser sedan av ett index ifrån sekvensen och slår upp det indexet i ordlistan, motsvarande symbol/symboler adderas till resultatet. Under tiden byggs ordlistan upp på samma sätt som den byggdes upp under komprimeringen. Det är dock lite annorlunda under avkomprimeringen, när man avläser en symbol, W, vet vi att den sista symbolen i W är ett prefix till nästa symbol, X, som ska vara i ordboken, så vi måste därför vänta tills vi läser av nästa symbol, Y, innan vi kan addera X till ordboken. Avkomprimeringen ligger på så vis ett steg efter med att bygga upp ordboken [3].

3 Metod

Nedan gås igenom hur entropi-estimering, Huffman-kodning och LZW-kodning har genomförts.

Entropi estimering

För att göra en entropi estimering av givna källor, som angavs i laborationens uppgift, skrevs ett program i C++. Programmet utgick ifrån en funktion som beräknade entropin för en given källa enligt formeln som beskrivs i teori avsnittet. Funktionen tar in en sannolikhetstabell som mappar varje symbol mot dess sannolikhet att den förekommer i källan. Denna sannolikhetstabell behövde alltså genereras innan entropin kunde beräknas.

För att generera en sannolikhetstabell estimerades sannolikheterna för varje symbol i källan. Detta gjordes genom att först spara hela filen i en lista, för att sedan iterera över den och addera 1 i en ny lista för varje påträffad symbol i källan. Denna nya lista mappar således symboler mot hur många gånger den förekommer i källan, det vill säga, en frekvenstabell. Denna frekvenstabell användes sedan för att beräkna sannolikhetstabellen genom att dividera varje frekvens med källans storlek. Den önskade entropin kunde sedan beräknas.

I laborationen har även betingade entropier beräknats för källorna. En betingad entropi betyder att vi beräknar entropi då vi känner till hur källan ser ut K steg tillbaka i tiden. De betingade entropierna som har beräknats är då K = 1 och då K = 2, detta utrycks genom följande, H(Xi | Xi-1) och H(Xi | Xi-1, Xi-2). För att beräkna de betingade entropierna krävdes en sannolikhetstabell för par av symboler och för tripplar av symboler. Dessa sannolikhetstabeller togs fram på motsvarande sätt som för en symbol, med undantag att två respektive tre symboler lästes in i taget. Dessa symboler adderades sedan genom att konkatenera dess bit representation, detta skapar en unik ny symbol. Vi har då sannolikhetstabeller för par och tripplar av symboler. Sedan beräknades den gemensamma entopin, H(X1, X2) och H(X1, X2, X3) med samma entropi funktion som användes för singlar av symboler. Efter det användes kedjeregeln, H(X1, X2, ..., Xn) = H(X1) + H(X2 | X1) + ... + H(Xn | X1, ..., Xn-1), för att beräkna den betingade entropin.

Huffman-kodning

För att komprimera godtyckliga filer med Huffman-kodning skrevs ett program i C++. Grundmetoden som användes var den som beskrevs i teori avsnittet. Först lästet filen in och varje symbol sparades i en lista. Sedan byggdes en frekvenstabell upp på samma vis som gjordes för att estimera entropin. Utifrån denna frekvenstabell byggdes sedan ett Huffman-träd, som beskrivs i teoriavsnittet. Detta Huffman-träd bestod av noder, structar i C++, som innehåller symbolen, frekvensen för symbolen och två stycken pekare till andra noder. För att bygga upp Huffman-trädet används en prioritetskö, det behövs för att noderna i kön behöver vara sortera med den lägsta frekvensen först. Sedan tas de två första noderna ut ur kön och deras frekvens summeras och en ny nod bildas med de två gamla noderna som barn till den nya noden. Den nya noden läggs sedan in i prioritetskön igen. Detta upprepas till bara en nod återstår i prioritetskön, denna nod är rotnoden i Huffman-trädet.

När Huffman-trädet är uppbyggt byggdes sedan en så kallad kodningskarta upp, det vill säga en lista som mappar symboler mot sekvenser av bitar. Detta gjordes genom att rekursivt traversera genom hela Huffman-trädet och varje gång vi gick till höger barn adderades en 1:a och varje gång vi gick till vänster barn adderades en 0:a.

Med denna kodningskarta kodades sedan källan. Detta gjordes genom att återigen öppna upp filen och gå igenom symbol för symbol. För varje symbol i filen adderades motsvarande sekvens av bitar i kodningskartan till en resulterande sträng. Denna sträng skrevs sedan till en ny fil och källan har därmed kodats med Huffman-kodning.

En viktig detalj att nämna här är att den frekvenstabell som genererades i början av Huffman-kodningen sparades i den kodade filen, som en så kallad header. Detta är ett måste eftersom vi behöver veta frekvenserna för varje symbol när vi ska avkomprimera den kodade Huffman-filen.

För att avkomprimera den komprimerade Huffman-filen läses först headern av och frekvenstabellen erhålls. Med denna frekvenstabell byggs sedan samma Huffman-träd upp som vid komprimering av källan. Sedan avkomprimeras Huffman-filen genom att börja från rotnoden och sedan läsa av bit för bit. Vid en 0:a går vi ner i vänstra subträdet och vid en 1:a går vi ner i högra subträdet. När vi stöter på en lövnod adderar vi den motsvarande symbolen till resultatsträngen och börjar om ifrån rotnoden. På så vis byggs orginalfilen upp igen.

LZW-kodning

För att komprimera godtyckliga filer med LZW-kodning skrevs ett program i C++. Grundmetoden som användes utgicks ifrån den som beskrevs i teori avsnittet. Först lästes den givna filen in och varje symbol sparades i en lista. Sedan byggdes ordboken upp, bestående av alla möjliga bytes, 0-255. Sedan itererades det över varje symbol den sparade filen, om den symbolen redan existerade i ordboken så lästes en nästa symbol in och konkateneras till den föregående symbolen. Tillexempel om A läses in och den symbolen redan existerar i ordboken läses nästa symbol B in och konkateneras till A, det ger resultatsymbolen AB. Sedan återupprepas proceduren och det undersöks om symbolen AB finns i ordboken eller inte.

Låt oss nu säga att symbolen AB inte finns i ordboken. Det betyder att vi ska lägga till AB i ordboken och skriva A:s motsvarande index i ordboken till resultatfilen. A:s index skrivs med X antal bitar där X är bas 2 logaritmen av nuvarande storleken på ordboken. Detta upprepas till det inte finns fler symboler kvar att läsa vilket resulterar i en komprimerad LZW-fil.

För att avkomprimera en komprimerad LZW-fil läses först LZW-filen in och varje bit sparades i en lista. Sedan byggdes samma motsvarande ordbok upp som vid komprimering av filen med undantaget att denna gång mappas index mot bytes. Sedan läser vi av X antal bitar ifrån listan där X är bas 2 logaritmen av nuvarande storleken på ordboken. Dessa byter transformeras till motsvarande siffra, med denna siffra slås motsvarande symbol upp i ordboken. Under tiden i avkomprimeringen byggs ordboken upp så att den blir identisk med den ord-

bok som användes vid komprimeringen av källan. Hur detta görs förklaras i teori avsnittet. På detta vis erhålls en identisk källa som innan komprimering.

4 Results

This chapter presents the results. Note that the results are presented factually, striving for objectivity as far as possible. The results shall not be analyzed, discussed or evaluated. This is left for the discussion chapter.

In case the method chapter has been divided into subheadings such as pre-study, implementation and evaluation, the result chapter should have the same sub-headings. This gives a clear structure and makes the chapter easier to write.

In case results are presented from a process (e.g. an implementation process), the main decisions made during the process must be clearly presented and justified. Normally, alternative attempts, etc, have already been described in the theory chapter, making it possible to refer to it as part of the justification.

5 Discussion

This chapter contains the following sub-headings.

5.1 Results

Are there anything in the results that stand out and need be analyzed and commented on? How do the results relate to the material covered in the theory chapter? What does the theory imply about the meaning of the results? For example, what does it mean that a certain system got a certain numeric value in a usability evaluation; how good or bad is it? Is there something in the results that is unexpected based on the literature review, or is everything as one would theoretically expect?

5.2 Method

This is where the applied method is discussed and criticized. Taking a self-critical stance to the method used is an important part of the scientific approach.

A study is rarely perfect. There are almost always things one could have done differently if the study could be repeated or with extra resources. Go through the most important limitations with your method and discuss potential consequences for the results. Connect back to the method theory presented in the theory chapter. Refer explicitly to relevant sources.

The discussion shall also demonstrate an awareness of methodological concepts such as replicability, reliability, and validity. The concept of replicability has already been discussed in the Method chapter (3). Reliability is a term for whether one can expect to get the same results if a study is repeated with the same method. A study with a high degree of reliability has a large probability of leading to similar results if repeated. The concept of validity is, somewhat simplified, concerned with whether a performed measurement actually measures what one thinks is being measured. A study with a high degree of validity thus has a high level of credibility. A discussion of these concepts must be transferred to the actual context of the study.

The method discussion shall also contain a paragraph of source criticism. This is where the authors' point of view on the use and selection of sources is described.

In certain contexts it may be the case that the most relevant information for the study is not to be found in scientific literature but rather with individual software developers and open

source projects. It must then be clearly stated that efforts have been made to gain access to this information, e.g. by direct communication with developers and/or through discussion forums, etc. Efforts must also be made to indicate the lack of relevant research literature. The precise manner of such investigations must be clearly specified in a method section. The paragraph on source criticism must critically discuss these approaches.

Usually however, there are always relevant related research. If not about the actual research questions, there is certainly important information about the domain under study.

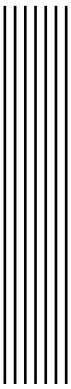
5.3 Källkritik

Wikipedia ska sagts inte användas som källor i en rapport men inom just datavetenskap är wikipedia oftast en trovärdig källa och har därför använts i denna rapport. En annan anledningen beror också på att just Huffman-kodning och LZW-kodning är kända metoder för komprimering av filer och för just fakta angående metoderna tycker jag att wikipedia är en pålitlig källa.

6 Conclusion

This chapter contains a summarization of the purpose and the research questions. To what extent has the aim been achieved, and what are the answers to the research questions?

The consequences for the target audience (and possibly for researchers and practitioners) must also be described. There should be a section on future work where ideas for continued work are described. If the conclusion chapter contains such a section, the ideas described therein must be concrete and well thought through.



Litteratur

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest och Clifford Stein. *Introduction to algorithms, 3rd edition*. 3. utg. MIT Press, 2009, s. 428–437.

[2] Harald Nautsch. *TSBK08 - Data compression*. 2018. URL: http://www.icg.isy.liu.se/en/courses/tsbk08/lect2.pdf.

[3] Terry Welch. "A Technique for High-Performance Data Compression". I: *Computer* 17.6 (1984), s. 8–19. DOI: 10.1109/mc.1984.1659158.