OU 1 - Huffman

Innehåll

Introduktion

Huffmankodning är en datakomprimeringsalgoritm som används av flera olika populära filformat som tex ZIP, JPEG och MP3 som en del av komprimeringsprocessen. Vid Huffmankodning så lagras de symboler som man vill lagra med en variabel längd i motsats till hur tex datatypen char lagras i c, där varje bokstav alltid upptar 8 bitar. Vid Huffmankodning så kodar man oftare förekommande symboler med en kortare kod, medan symboler som förekommer mindre ofta kodas med längre bitsekvenser. En noggrann genomgång av algoritmen för Huffmankodning finns i lektionsmaterialet.

Syfte

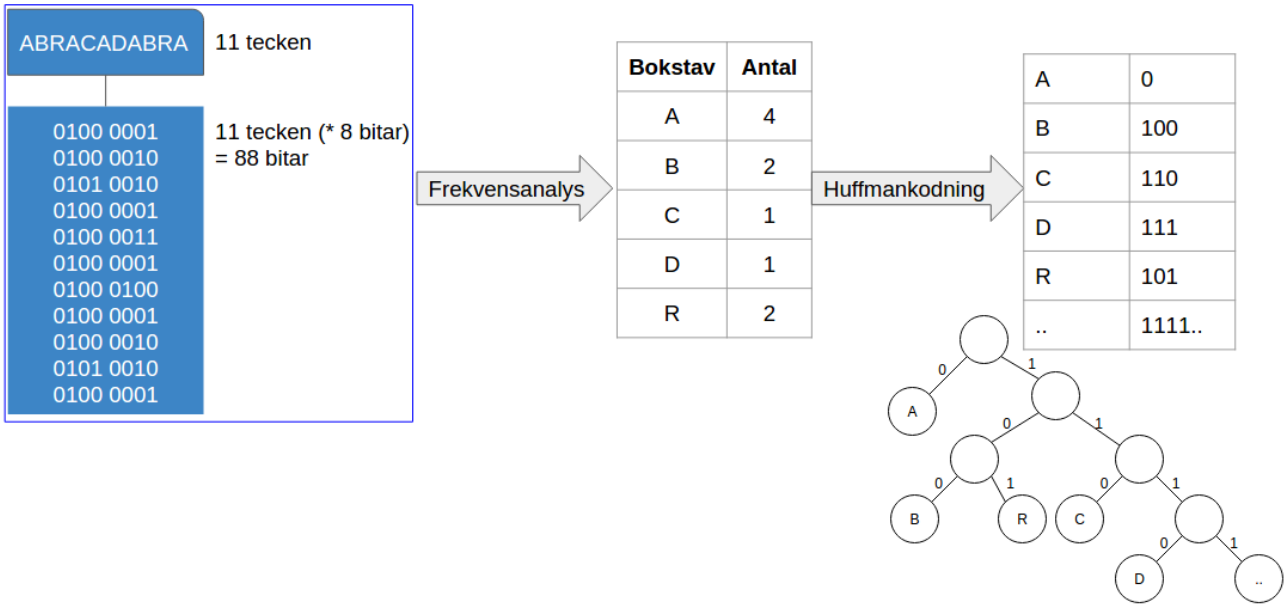
Syftet med denna obligatoriska uppgift är att du ska få öva på datatypen träd, få en förståelse för hur olika sätt att lagra och överföra data kan påverka ett program samt öva på att skriva program som man interagerar med via kommandoraden. Uppgiften är direkt kopplad till följande förväntade studieresultat:

- beskriva och använda sig av abstrakta datatyper och algoritmer (FSR 1),
- implementera lösningen på ett problem i form av ett program i programspråket C inklusive att välja en lämplig representation för de ingående abstrakta datatyperna (FSR 7),
- utifrån en frågeställning göra relevanta designbeslut under arbetet med att lösa ett problem utifrån sin kunskap om hur struktur-, tids- och rumsaspekter påverkar kvalitet hos program (FSR 9).

Uppgiften ska utföras **enskilt**.

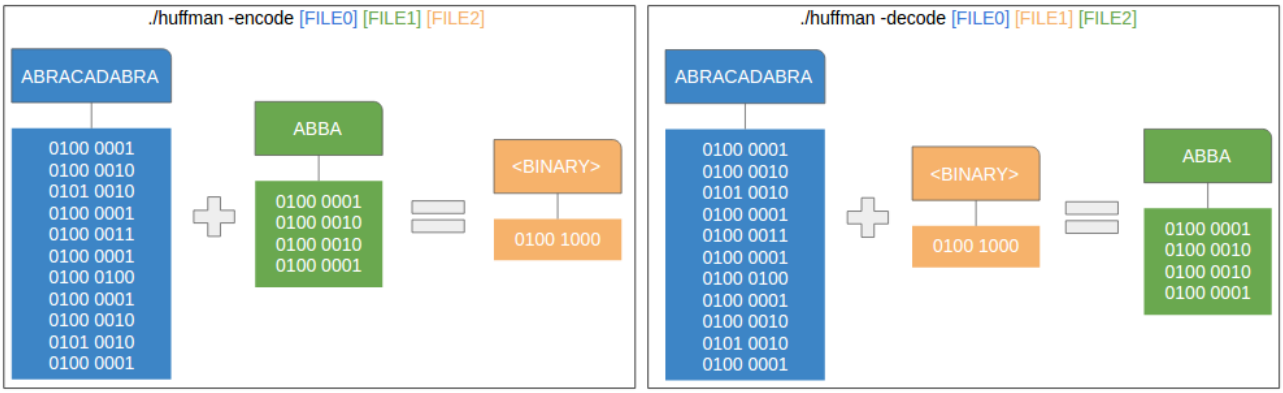
Bakgrund

I denna obligatoriska uppgift så ska du skriva ett program som läser in en fil med text i (där varje tecken lagras i en byte) och gör en frekvensanalys på filen. Resultatet från frekvensanalysen ska du sedan använda för att, med hjälp av Huffmankodning, ta fram en tabell över tecken och deras bitrepresentation. Kom ihåg att alla 256 tänkbara tecken måste finnas med i tabellen, inte bara de som finns i in-filen. För ett exempel se figur 1.



Figur 1: Exempel på Huffmankodning.

Notera att samtliga möjliga tecken (".." symboliserar dessa tecken i exemplet i figur 1) måste finnas i frekvensanalysen även om de inte förekommer i filen som analyseras. Resultatet från frekvensanalysen kan sen användas antingen för att komprimera innehållet i en fil och spara resultatet i en annan fil eller för att avkoda en redan komprimerad fil (se figur 2).



Figur 2: Exempel på kodning av fil och avkodning av fil med hjälp av Huffmankodning.

Genomförande

Ditt färdiga program ska köras från prompten (dvs utanför en IDE som tex CodeLite), ska heta **`huffman`** och fungera så här:

```
USAGE:

huffman [OPTION] [FILE0] [FILE1] [FILE2]

Options:

-encode encodes FILE1 according to frequence analysis done on FILE0. Stores the result in FILE2

-decode decodes FILE1 according to frequence analysis done on FILE0. Stores the result in FILE2
```

Dvs man anropar programmet med en option som talar om ifall infilen ska kodas eller avkodas. Sen följer tre filnamn. Den första filen används för att skapa frekvensanalysen, den andra är filen som ska kodas eller avkodas och den sista är namnet på filen som resultatet ska lagras i. Den sista filen existerar inte utan skapas av programmet.

Exempel på hur testkörningar av programmet kan se ut

Exempel 1

Frekvensanalys görs på "frequence.txt", sen kodas filen "infil.txt" och resultatet lagras i "utfil.txt" (se vänstra delen i figur 2). Notera att programmet också ska ge information om hur många bytes som används. Eftersom filer kan lagras i olika teckenkodningar kan siffrorna variera för samma fil beroende på hur man sparat ned den (tex val av browser kan påverka). Utskriften måste se exakt ut som nedanstående (siffrorna och filnamnet kan så klart variera!):

```
> huffman -encode frequence.txt infil.txt utfil.txt

6489 bytes read from infil.txt.

4456 bytes used in encoded form.
```

Exempel 2

Frekvensanalys görs på "frequence.txt", sen avkodas filen "infil.txt" och resultatet lagras i "utfil.txt" "(se högra delen i figur 2). Utskriften ska se ut exakt enligt nedan (filnamn kan variera):

```
> huffman -decode frequence.txt infil.txt utfil.txt

File decoded succesfully.
```

Exempel 3:

Körning av programmet utan parametrar eller på annat felaktigt sätt ska ge en hjälptext om hur programmet körs. Om felen som uppkommer beror på att felaktiga filnamn angetts ska detta också ge relevanta felmeddelanden. För exempel på utskrift se nedan (i detta fall ska utskriften se exakt ut så som visas):

```
> huffman

USAGE:

huffman [OPTION] [FILE0] [FILE1] [FILE2]

Options:

-encode encodes FILE1 according to frequence analysis done on FILE0. Stores the result in FILE2

-decode decodes FILE1 according to frequence analysis done on FILE0. Stores the result in FILE2
```

Tips och hjälpmedel

OBS! Alla filer som finns inlänkade i detta avsnitt finns också samlade i en zip-fil längst ned i dokumentet.

Planera innan du börjar koda

Att planera före du börjar skriva ditt program är kanske det enskilt viktigaste rådet. Frågor du bör fundera över kan vara: Vilka delproblem finns i uppgiften? Finns det delproblem som kan implementeras och testas oberoende av de andra delarna (till

exempel felhanteringen av uppstarten av programmet om man anger fel antal parametrar eller filer som inte finns)? Vilka datatyper behöver jag använda? Vilka datatyper finns givna och vilka måste jag skriva själv? Hur ska strukturen på programmet se ut? Dels algoritmmässigt men också filmässigt. Ditt färdiga program kommer bestå av en större mängd programfiler än vanligt, exempelvis ska alla datatyper ligga i egna .c och .h-filer.

Teckenhantering

Alla tecken som finns i filen ska tas med vid kodningen (alla specialtecken osv). Sist i datat som du ska koda (efter datat från filen) ska du placera in ett *End of transmission*-tecken (EOT). I C kan EOT skrivas som '\4'. På så vis så vet du att när du avkodar ditt data och kommer till ett sådant tecken så kan du sluta avkodningen. Har man inget sluttecken kan man annars få med skräp vid avkodning på grund av att datat i en fil kommer behöva sparas i hela bytes. För att detta inte ska ge större påverkan än nödvändigt på storleken av det komprimerade datat så ska du också räkna med ett (1) sådant tecken då du beräknar teckenfrekvensen från en fil.

Datatyper

I detta avsnitt ger vi länkar till ett antal datatyper som vi tror du har nytta av. Dessa får du använda fritt i din lösning (med kommentarer i filerna om vem som ursprungligen skrivit dem). Du får också modifiera dem så att de passar din lösning bättre om det behövs men då ska det beskrivas i både kommentarer och rapport. Alla datatyper är samlade i katalogen Datatyper i filsamlingen och ingår också i zip-filen längst ned i detta dokument. För att inte göra detta dokument för stort återger vi inte gränsytan för de datatyper som är "standard". Dessa finns i h-filerna till datatypen. För de flesta finns också ett litet program som visar hur datatypen kan användas.

NOTERA:

- Filerna får endast användas under denna kurs. Om du vill använda delar av dem på kommande kurser krävs speciellt tillstånd från den som skrivit dem!
- Du kommer behöva komplettera dessa datatyper med egna datatyper. Exempelvis datatyper för:
  - frekvenstabellen och tabellen för kodning/avkodning mellan huffmankod och vanliga tecken (du får återanvända hela eller delar av dina egna lösningar på OU3 på DV1-kursen).
  - triet som byggs upp. När du skapar denna datatyp så tänk på att den troligen fungerar bättre i din lösning om den är delträdsorienterad och inte navigeringsorienterad. Detta eftersom den kommer behöva operationer för att slå samman två trie till ett nytt trie och samtidigt beräkna värdet som finns i roten utifrån delträdens värden.

Prioritetskö

För varje varv i kodningen ska man plocka ut de för tillfället två minsta träden och slå ihop dem. Ett enkelt sätt att göra detta är att använda sig av en prioritetskö som lagrar träden och där prioriteten är vikten för trädet. När man slagit ihop träden läggs det nya trädet in i kön. Fundera på hur funktionen för jämförelse av prioriteter ska skrivas för att det slutliga trädet ska ha så låg höjd som möjligt.

Lista

Används av prioritetskön. Denna variant är en dubbellänkad lista.

BitSet

När man skapat ett Huffmanträd kan det vara bra att skapa en tabell där man har tecknet som nyckel och dess motsvarande bitmängd som värde. Eftersom det inte finns en inbyggd datatyp i c som hanterar bitmängder av varierande storlekar har vi implementerat en åt dig. Tillsammans med själva datatypen finns också en fil test_bitset.c som visar ett exempel på hur man kan använda det. Gränsytan för bitset ser ut så här (mer utförliga kommentarer finns i h-filen):

bitset *bitset_empty();

Creates a new empty bitset of length 0 and returns it

void bitset_setBitValue(bitset *b, int bitNo, bool value);

Set bit bitno to value (true=1, false=0) in the bitset b. bitNo should be >= 0.

If bitNo >= bitset_size(b) the bitset will be extended up to that bit.

int bitset_size(bitset *b);

Gives the size of the bitset b.

bool bitset_memberOf(bitset *b, int bitNo);

Returns the value of the given bitNo within the bitset.

Not defined if bitNo >= bitset_size(b)

char *bitset_toByteArray(bitset *b);

Converts the bitset to an array of bytes. The array will contain all bits existing within the array.

If the bitset_size is not a multiple of 8 bits, the last byte of the array will be padded with bits of value 0.

void bitset_free(bitset *b);

Deallocates all memory used by the given bitset.

Filer att testa ditt program på

Egentligen fungerar det med vilka filer som helst men för att få ett exempel att kontrollera så ger vi några testfiler här.

- löremipsum.txt - en svensk "nonsenstext" som är genererad via sidan http://löremipsum.se//#/
- balenPaEkeby_GostaBerlingsSaga_SelmaLagerlof.txt - kaptitlet Balen på Ekeby från Gösta Berlings saga skriven av Selma Lagerlöf. Texten hämtad från projekt Runeberg http://runeberg.org/

Om man använder löremipsum.txt för att bygga upp frekvenstabellen och sen kodar Selma Lagerlöfs text ska filen före komprimering vara på ca 33983 bytes och efter kodning ska den vara ca 23124 bytes (kan variera lite beroende på hur webbläsare hanterar filen teckenkodning när du laddar ner filen).

Inlämning

Uppgiften bedöms dels som godkänd eller inte godkänd och ges också ett sifferomdöme i intervallet 0-10. Notera att den första gjorda inlämningen bedöms (så lämna inte in halvfärdiga uppgifter om du vill få höga poäng). Vid kompletteringsdatum ges omdömen i intervallet 0-8 poäng och vid uppsamlingsdatumet ges omdömen i intervallet 0-6 poäng (för att reflektera det faktum att du fått mer tid till att jobba med uppgiften).

- Den obligatoriska uppgiften ska genomföras **enskilt** och vara inlämnad via Labres (källkod och rapport) **senast 17.00 den 8/2 2018.**
- Uppgiften ska redovisas via en skriftlig rapport **i pdf-format** och källkodsfiler.
- Rapporten ska vara skriven med den som bedömer uppgiften som målgrupp. Det innebär att den som läser din rapport är fullt insatt i vad som ska göras och har de teoretiska kunskaperna som krävs för att kunna bedöma lösningen av uppgiften. Rapporten ska innehålla följande:
  - Framsida som innehåller kursens namn, ditt namn, cs-id, umu-id och uppgiftens nummer.
  - Ett avsnitt som beskriver hur ditt program ska kompileras, vilka filer som berörs samt hur programmet exekveras och vilka in- (argument/filer) och utdata (terminal/filer) som kan förväntas.

- En beskrivning, gärna med tillhörande relevanta figur till stöd för läsaren, som visar hur strukturen på ditt program är upplagt. Man ska kunna utläsa vilka datatyper som du använt från vår samling, vilka datatyper du har skrivit själv och hur dessa hänger ihop.

- Källkoden
  - Välstrukturerad och kommenterad kod lämnas in i **separata** filer. Huvudprogrammet som knyter samman de olika delarna ska ligga i **huffman.c**.
    - OBS! Lämna endast in .c och .h-filer, lämna inte in objektfiler skapade av kompilatorn eller körbara filer.
  - Programmet ska fungera och inte läcka minne. De tre exempelkörningarna som nämns under Genomförande ska fungera som specificerat och alla kodade filer som genereras ska innehålla ett "end-of-transmission tecken" (se Tips och hjälpmedel).
  - Koden ska kunna kompileras med gcc -std=c99 -Wall utan varningar/felmeddelanden.

Bedömning

Bedömning sker på den första inlämningen av uppgiften så se till att den är så bra som möjligt! För betyget G krävs att samtliga kriterier har bedömts som godkända och erhållna poäng vägs samman med övriga uppgifter till ett totalbetyg på momentet endast när samtliga uppgifter har omdömet G.

Saker som vi kommer titta på i samband med poängsättningen av uppgiften är:

- Om programmet fungerar enligt specifikation, har god felhantering och en så bra rumskomplexitet som möjligt.
- Om programmet är korrekt indenterat, har bra namngivning av funktioner och variabler, har väl kommenterade funktioner och att kommentarerna (både till funktioner och eventuella kommentarer i kod) är enhetligt skrivna på en bra nivå i förhållande till koden.
- Om programmet använder sig av god abstraktion, har bra struktur, har en lämplig uppdelning i funktioner och en bra användning av språkets konstruktioner.
- Om programmet är fritt från minnesfel, inklusive minnesläckor, korruption och ogiltiga pekare.
- Om rapporten har en välbyggd disposition och en tydlig målgrupp. Rapporten är lättläst och tydligt. Eventuella bilder och tabeller har korrekta bildtexter och referenser till dessa och/eller andra källor är inkluderade på ett korrekt sätt.

Program som inte fungerar eller inte går att läsa (något mindre fel kan tolereras) eller rapporter som saknar delar eller är bristfälliga kommer att leda till att man måste komplettera uppgiften. Om en uppgift måste kompletteras påverkar naturligtvis detta poängtalet negativt.

Givna filer

Zip-fil med samtliga givna filer