# 5DV204/VT18: OU1 – Huffman

## Buster Hultgren Wärn

## [dv17bhn@cs.umu.se](mailto:dv17bhn@cs.umu.se)

## 2018-02-15

# Innehåll

1. Inledning – s. 2
2. Experimentell analys – s. 2  
   2.1. Experiment – s. 2  
   2.2. Resultat och analys – s. 2
3. Asymptotisk analys – s. 5  
   3.1. Analys av det bästa fallet – s. 5  
   3.2. Analys av det värsta fallet – s. 5
4. Referenser – s. 6

# Inledning

Denna uppgift gick ut på att skriva ett komprimerings program som kan komprimera och även läsa komprimerade filer. Programmet gör först en frekvensanalys på en textfil, vilket innebär hur ofta olika karaktärer i texten kommer upp. Desto oftare, desto högre vikt får karaktären. Efter detta byggs en trie upp, där noder med högre vikt hamnar närmare roten.

Sökvägen till varje löv, representeras som en sträng av ettor och nollor, och de karaktärer som har störst vikt får kortast sökväg. Då en vanlig char i C består av åtta bitar så kan sökvägen bestå av en sträng på färre ”bitar”, och därmed kan textfilen komprimeras.

# Givna datatyper

Till uppgiften gavs tre stycken givna datatyper. En Dubbelriktad lista, en prioritetskö och en bithanterare – bitset. Listan användes endast i prioritetskön, och prioritetskön användes för att bygga noderna i trädet till.

Bitset är en datatyp för att hantera bitar. Bitar kan läggas in i den, som konverteras till en char array där varje char representerar en hel byte. Det är denna array med chars som sedan skrivs till filen, för att få den komprimerade filen. Däremot var denna datatyp problematisk, och användes inte i implementationen.

# Implementation

**Hufftree.c**

Hufftree.c är en datatyp specifikt gjord för att bygga ett huffman träd. Datatypen är uppdelad i två huvudsakliga områden. Själva trädet, och dess noder.

Trädet består av en Huffman tabell och en rot. Därefter har trädet sex olika operationer vilka är:

Empty (root, size): Skapar ett nytt träd med roten ”root” och storleken ”size”. Returnerar trädet.  
IsEmpty (tree): Undersöker ifall ett träd ”tree” är tomt, returnerar i sådana fall 1, annars 0.  
Kill (tree): avallokerar allt minne som trädet ”tree” har allokerat. Returnerar ingenting.  
GetRoot (tree): Returnerar roten för trädet ”tree”.  
Traverse (tree): Traverserar trädet “tree” för att bygga upp Huffman tabellen.   
GetKeyPath (tree, key): Returnerar

**Huffman.c**

Huffman.c är ”main” filen som kör alla andra olika filer. Den sköter hand om programmets flöde, filvalidering och frekvensanalys.

**Encode.c**

Encode.c sköter hand om krypteringens flöde. Detta är efter det att ett Huffman träd har skapats, och kommandot ”-encode” har används som parameter för programmet.

**Decode.c**

Decode.c är flödeshanteringen då programmet ska läsa en krypterad fil och ”avkoda” denna fil. Detta sker efter att Huffman trädet har skapats och kommandot ”-decode” har används som parameter för programmet.

**Analys**

Ett antal olika delar behövde implementeras för att programmet skulle fungera. Dessa var filvalidering, frekvensanalys, Huffman träd, kryptering och läsning av kryptering. Allt detta skulle även knytas ihop så att de fungerade med varandra. Detta gjordes i huffman.c filen, som kallar på funktioner och styr flödet av programmet.

Enocde.c och Decode.c är uppfölare på huffman.c, beroende på ifall en fil ska krypteras eller en krypterad fil ska läsas.

Eftersom bitset datatypen inte används så behövdes en annan datatyp för att hantera bitar implementeras. Denna är bitString.

**Huffman.c**

Huffman.c hanterar flödet av programmet. Först och främst görs filvalidering och en frekvensanalys. Med frekvensanalysen byggs en prioritets kö upp, som hanterar noder. Desto oftare en karaktär används i frekvensanalysen, desto större vikt får noden med denna nyckel. Noder med minst vikt har högre prioritet i prioritetskön. Detta är så att kön ska bygga upp ett träd genom att slå ihop två noder till en nod som får dess båda

**HuffTree.c**

HuffTree är ett binärt sökträd, också en trie, byggd specifikt för Huffman komprimering. Trädet kan byggas ut både fullt eller komplett, beroende på användning. I detta program så byggdes det komplett. Varje nod/löv som skapas har en vikt och en nyckel – vilket är en unsigned char för att kunna representera extended ascii.

**BitString.c**

**Encode.c**

**Decode.c**

# Slutsats