# DM507 Algoritmer og datastrukturer

Forår 2016

Projekt, del III

Institut for matematik og datalogi
Syddansk Universitet

20. april, 2016

Dette projekt udleveres i tre dele. Hver del har sin deadline, således at arbejdet strækkes over hele semesteret. Deadline for del III er fredag den 13. maj. Projektet skal besvares i grupper af størrelse to (evt. tre, efter aftale med underviser).

## Mål

Målet for del III af projektet er at komprimere filer via Huffman-kodning. Der skal laves to programmer: et til at kode/komprimere en fil, og et til dekode den igen.

Vær sikker på at du forstår Huffmans algoritme (Cormen et al. afsnit 16.3 indtil side 433) før du læser resten af denne opgavetekst.

## Opgaver

### Opgave 1

Der skal i Java implementeres et program, der læser en fil og laver en Huffman-kodet version af den. Den præcise definition af input og output følger nedenfor. Programmet skal hedde `Encode.java`, og skal kunne kaldes sådan:

```
Java Encode nameOfOriginalFile nameOfCompressedFile
```

Input-filen skal ses som en sekvens af bytes (8 bits), dvs. der er $2^8 = 256$ mulige forskellige tegn i input. Tegn kaldes i resten af opgaven for bytes. For at læse bytes fra en fil, skal man bruge `read`-metoden fra `FileInputStream` (eller evt. `BufferedInputStream`). Man skal ikke bruge en stream fra `Reader`-

familien. I Java repræsenteres bytes ved `int`'s, hvor man kun bruger værdierne 0 til 255,[1] og dette er typen af output for `read`-metoden fra `FileInputStream` og input for `write`-metoden fra `FileOutputStream`. En byte er derfor otte bits i input- og outputfilen på disk, men er en `int` undervejs i Java-programmet.

Programmet `Encode` skal virke således:

1. Scan inputfilen og lav en tabel (et array) over hyppigheden af de enkelte bytes (som altså er `int`'s mellem 0 og 255).
2. Kør Huffmans algoritme med tabellen som input. Det er nok at køre den på bytes med hyppighed større end nul.
3. Scan inputfilen igen, og konverter denne gang hver byte til sit kodeord (en sekvens af bits), og skriv disse bits til outputfilen.

Under punkt 3 skal man konvertere bytes fra input til kodeord for output. Dette kan f.eks. ske ved at man i starten laver et rekursivt gennemløb (svarende til et inorder-gennemløb af et søgetræ) af Huffman-træet og genererer alle kodeordene én gang for alle. Under gennemløbet skal man hele tiden vedligeholde et kodeord svarende til stien fra roden til den nuværende knude, og når man når et blad, kan dette kodeord gemmes i en tabel (et array af længde 256) af kodeord. For hver byte i input kan man derefter bare slå op i tabellen. Man må her vælge en brugbar repræsentation af kodeord (som jo ikke er lige lange). Én mulig repræsentation er, at et kodeord gemmes som en `String` af `'0'` og `'1'` tegn. En sådan streng kan så under et opslag i tabellen gennemløbes tegn for tegn og konverteres til bits til output.

I ovenstående scannes inputfilen flere gange. Dette er at foretrække frem for at scanne den én gang og derefter gemme dens indhold i et array til videre brug, eftersom RAM-forbruget derved stiger fra $O(1)$ til inputfilens størrelse (som kan være meget stor).

Huffman-kodning skal implementeres via en prioritetskø (jvf. Cormen et al. side 431). Hertil skal genbruges interfacet `PQ` fra del I, samt gruppens implementation `PQHeap` heraf. Også klassen `Element` skal genbruges, og skal her repræsentere deltræer genereret under Huffman-algoritmens kørsel. Derfor skal `data` i `Element` være et træ-objekt, og `key` skal være dets tilhørende hyppighed. Træ-objektet gemt i `data`-delen er et binært træ med hyppigheder i alle knuder og derudover med en byte (`int` med værdi mellem 0 og 255) i blade. Se figuren i bogen side 432. Træ-objekterne fra del I kan *ikke* direkte genbruges (vi skal f.eks. her bruge andre operationer på dem end søgetræsoperationer), men kan tjene som inspiration.

[1] Bemærk at der findes en datatype `byte` i Java, men denne er et signed 8-bit heltal i two's complement og har derfor ikke de rette egenskaber, så den skal *ikke* anvendes.

Den præcise specifikation af output er:

> Først 256 `int`'s (hvilket fylder $256 \cdot 32$ bits i alt), som angiver hyppighederne af de 256 mulige bytes i input, derefter de bits, som Huffman-kodningen af input giver.

Bemærk at for korte filer (eller lange filer, som ikke kan komprimeres væsentligt med Huffmans metode) kan output af `Encode` være længere end den oprindelige fil. Man kan tænke sig mange måder at undgå eller begrænse denne situation på, men dette er ikke en del af projektet.

Når man skriver et kodeord i output, har man brug for at skrive bits én ad gangen. Der er i Javas bibliotek ikke metoder til at læse og skrive enkelte bits til disk (mindste enhed er en byte), men underviseren har udleveret et bibliotek, som kan gøre dette. I dette er der også mulighed for at læse og skrive hele `int`'s (disse fylder 32 bits).

Bemærk: I opgave 1 skal man bruge `read`-metoden fra `FileInputStream` til at læse bytes fra inputfilen (den originale fil). Man skal bruge metoderne fra det udleverede bibliotek til at skrive `int`'s (for hyppighedstabel) og bits (for Huffmans-koderne) til outputfilen (den komprimerede fil).

## Opgave 2

Der skal i Java implementeres et program, som læser en fil med data genereret af programmet fra opgave 1, og som skriver en fil med dens originale (ukomprimerede) indhold. Programmet skal hedde `Decode.java`, og skal kunne kaldes sådan:

```
Java Decode nameOfCompressedFile nameForDecodedFile
```

Programmet skal virke således:

1. Indlæse tabellen over hyppighederne for de 256 bytes.
2. Generere samme Huffman-træ som programmet fra opgave 1 (dvs. *samme* implementation skal bruges begge steder, så der i situationer, hvor Huffman-algoritmen har flere valgmuligheder, vælges det samme).
3. Bruge dette Huffman-træ til at dekode resten af bits i input, og som output skrive den originale version af filen.

Alt dette kan gøres i ét scan af input.

Bemærk at det samlede antal bits fra udskrivningen af Huffman-koderne af det udleverede bibliotek bliver rundet op til et multiplum af otte (dvs. til et helt antal bytes), ved at nul-bits tilføjes til sidst når filen lukkes. Dette skyldes at man på computere kun kan gemme filer, som indeholder et sådant antal bits. Disse tilføjede bits må ikke blive forsøgt dekodet, da ekstra bytes så kan opstå i output. Derfor må man under dekodning finde det samlede antal bytes i originalfilen ved at summere hyppighederne, og under rekonstruktionen af den ukomprimerede fil holde styr på hvor mange bytes, man har skrevet.

For at skrive bytes til en fil, skal man bruge `write`-metoden fra `FileOutputStream` (eller evt. `BufferedOutputStream`). Man skal ikke bruge en stream fra `Writer`-familien.

Bemærk: I opgave 2 skal man bruge metoderne fra det udleverede bibliotek til at læse `int`'s (for hyppighedstabel) og bits (for Huffmans-koderne) fra inputfilen (den komprimerede fil). Man skal bruge `write`-metoden fra `FileOutputStream` til at skrive bytes til outputfilen (den genskabte originale fil).

## Formalia

Du skal kun aflevere dine Java source-filer. Disse skal indeholde grundige kommentarer. De skal også indeholde navnene og SDU-logins på gruppens medlemmer. Husk at levere alle nødvendige filer med, også f.eks. `PQHeap.java`, `PQ.java` og `Element.java` fra del I.

Dine programmer vil blive testet med mange typer filer (`txt`, `.doc`, `.jpg`,...), og du bør selv gøre dette inden aflevering, men du skal ikke dokumentere denne test.

Filerne skal afleveres elektronisk i Blackboard med værktøjet "SDU Assignment", som findes i menuen til venstre på kursussiden i Blackboard. De skal enten afleveres som individuelle filer eller som ét `zip`-arkiv (med alle filer på topniveau, dvs. uden nogen directory struktur). Der behøves kun afleveres under én persons navn i gruppen.

Filerne skal *også* afleveres udprintet på papir i instruktorens boks på Imada (vælg én af instruktorerne, hvis personerne i gruppen går på forskellige hold). Spørg Imadas sekretær hvis man ikke ved, hvor instruktorboksene er. Der skal blot afleveres én kopi per gruppe.

Det er *vigtigt* at overholde alle syntaktiske regler ovenfor (navngivning af klasser, kald af programmer, `zip`-arkiv uden directory struktur, ingen `package` statements), da programmerne vil blive testet på automatiseret måde.

Aflever materialet senest:

**Fredag den 13. maj, 2016, kl. 12:00.**

Bemærk at aflevering af andres kode eller tekst, hvad enten kopieret fra medstuderende, fra nettet, eller på andre måder, er eksamenssnyd, og vil blive behandlet som sådan. Man lærer desuden heller ikke noget.