DM507 Algoritmer og datastrukturer

Forår 2015

Projekt, del II

Institut for matematik og datalogi
Syddansk Universitet

21. april, 2015

Dette projekt udleveres i to dele. Hver del har sin deadline, således at afleveringerne, og dermed arbejdet, strækkes over hele semesteret. Deadline for del II er onsdag den 20. maj. Projektet skal besvares i grupper af størrelse to (evt. tre, efter aftale med underviser).

Mål

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

Vær sikker på at du forstår Huffmans algoritme (Cormen et al. afsnit 16.3) 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 tegn i input. For at læse bytes fra en fil, skal man bruge en `FileInputStream` (eller evt. en `BufferedInputStream`), ikke en stream fra `Reader`-familien. 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. I stedet anvendes ofte en `int`, hvor man kun bruger værdierne 0 til 255. Dette er tilfældet for `read`-metoden fra `FileInputStream`, og for de tilsvarende output-metoder.

Programmet `Encode` skal virke således:

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

Der er mange måder at konvertere input-bytes til output-koder. Man kan f.eks. have et array af længde 256, som peger på Huffmantræets blade, og for hver byte løbe stien mod roden igennem, mens man for hvert skridt på stien checker om man kom fra et venstre- eller højrebarn. Man kan også én gang for alle via et gennemløb af træet konverterer dette til en tabel (array af længde 256) af kodeord - hvor man må vælge en brugbar repræsentation af kodeord (som jo ikke er lige lange). Én mulighed for repræsentation er, at et kodeord gemmes som en `String` af `'0'` og `'1'` tegn, der så kan gennemløbes 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 er et binært træ med hyppigheder i alle knuder og derudover med tegn (`int` med værdi mellem 0 og 255) i blade. Træobjekterne fra del I kan ikke direkte genbruges (vi skal f.eks. her bruge andre operationer på dem end søgetræsoperationer), men kan tjene som inspiration.

Der er i Javas bibliotek ikke metoder til at læse og skrive enkelte bits (mindste enhed er en byte), men underviseren vil udlevere metoder, som kan gøre dette.

Den præcise specifikation af output er: Først 256 `int`'s (dvs. 1024 bytes i alt), som angiver hyppighederne af de 256 tegn, 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.

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 starte med at indlæse tabellen over hyppighederne for de 256 tegn. Det skal derefter generere samme Huffman-træ som programmet fra opgave 1 (dvs. samme implementation skal bruges, så der i situationer, hvor Huffman-algoritmen har flere valgmuligheder, vælges det samme), og skal slutteligt bruge det 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 Huffman-kodningens bits er blevet rundet op til et multiplum af 8 (dvs. til et helt antal bytes) af de udleverede metoder, ved at de har tilføjet nul-bits til sidst (dette skyldes at man på computere kun kan gemme filer, som indeholder et helt antal bytes). Disse tilføjede bits ikke må blive forsøgt dekodet (så kan et ekstra tegn opstå i output). Derfor må man under dekodning finde det samlede antal tegn ved at summere hyppighederne, og under rekonstruktionen af den ukomprimerede fil holde styr på hvor mange tegn, man har skrevet.

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

Formalia

Lav en rapport, som beskriver dine løsninger af opgaverne ovenfor. Da besvarelsen i store træk består af kode, skal rapporten struktureres som følger:

- En kort introduktion, som giver overblik over, hvad rapporten indeholder og besvarer.

- Din kode (evt. undtagen de mest trivielle dele), indlejret mellem tekstdele, som forklarer hvad koden indeholder og gør. Hvis du har nogle interessante designvalg (dvs. nogle, der ikke er eksplicit beskrevet i opgaveteksten), så beskriv disse i størst detalje. For at indlejre kode(bidder) i tekst i LaTeX kan du f.eks. bruge pakken `listings`, eller blot environment'et `verbatim`.

- Et afsnit med beskrivelse af de test, du har udført for at sikre dig at koden fungerer korrekt. Fokuser mest på, hvad du har valgt at teste, mindre på at dokumentere resultatet, sålænge dette er som forventet. Hvis der er måder, du er bekendt med, hvorpå programmet ikke løser den stillede opgave, skal du skrive dette eksplicit.

Husk at skrive navnene på personerne i gruppen på forsiden af rapporten.

Rapporten skal afleveres udprintet i instruktorens boks på Imada (vælg én af instruktorerne, hvis personerne i gruppen går på forskellige hold).

Derudover skal rapporten (i *pdf*-format) samt alle Java source-filer 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).

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

Aflever materialet senest:

Onsdag den 20. maj, 2015, kl. 12:00.