Institut for Matematik og Datalogi
Syddansk Universitet, Odense

4. april 2019
RF

DM507 – Opgaver uge 15

Eksaminatorier I

1. Cormen et al. opgave 2.3 (side 41). Hint: for at overskue situationen, prøv algoritmen på konkrete instanser, og skriv summen i spørgsmål **c** ud.
2. (∗) Cormen et al. øvelse 4.2-7 (side 83). Produktet af to komplekse tal $a + bi$ og $c + di$ er $(ac - bd) + (ad + bc)i$, dvs. opgaven består i at beregne $ac - bd$ og $ad + bc$ ud fra a, b, c og d, men ved kun at bruge tre multiplikationer. Svaret ligner lidt Strassens metode (men er simplere).
3. Cormen et al. øvelse 15.1-3 (side 370).
4. Cormen et al. øvelse 15.4-1 (side 396).
5. Cormen et al. øvelse 15.4-2 (side 396).
6. Eksamen juni 2010, opgave 4.
7. Eventuelle manglende opgaver fra sidste uge.

Eksaminatorier II

1. Opvarming til projektet del III: En byte (en gruppe på otte bits) er den mindste enhed af data, som CPU'er kan håndtere. En fil er blot en række af bytes, og indeholder derfor altid et multiplum af otte bits. Der er $2^8 = 256$ muligheder for en byte's indhold: 000, 001, 010, 011,..., 111. I Java kan `read()`-metoden fra Java-bibliotekets `FileInputStream` læse bytes fra en fil én ad gangen. 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`.

[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 nogle egenskaber, som ikke gør den så brugbar her.

I denne opgave skal du ved hjælp af `read()`-metoden fra Java-bibliotekets `FileInputStream` lave et Java-program som: i) læser en fil byte for byte, ii) undervejs i et array tæller, hvor mange af hver af de 256 mulige bytes filen indeholder, og iii) til sidst udskriver en tabel i stil med:

```
Byte 0: 0
Byte 1: 0
.
.
Byte 97: 7
Byte 98: 4
.
.
Byte 255: 0
```

Prøv programmet på nogle simple `.txt`-filer (uden danske bogstaver). Find på nettet en tabel over ASCII-koden, og brug den til at checke output. Prøv også programmet på andet end tekstfiler, f.eks. `.jpg`-filer og `.doc`-filer.

2. Opvarming til projektet del III: På websiden er lagt en Java-klasse `BitInputStream`, som tillader at vi kan tilgå de enkelte bits i en fil. Den virker ved at læse en byte forud, og fra denne pille de enkelt bits ud. Man behøver *ikke* forstå virkemåden for at kunne bruge klassen. Vi vil i denne opgave og den næste træne at bruge de offentlige metoder i klassen.

   I denne opgave skal du lave et Java-program som via kald til `readBit()`-metoden fra den udleverede `BitInputStream` læser de enkelte bits i en input `.txt`-fil én efter én, og undervejs udskriver disse bits som `'0'` og `'1'` tegn. Brug en tabel over ASCII-koden for at checke output fra en tekst `.txt`-fil.

3. Opvarming til projektet del III: En `int` i Java fylder fire bytes. Udover at kunne læse enkelte bits, kan `BitInputStream` også læse fire bytes i træk fra en fil og returnere dem som en `int` (som kan antage alle værdier mellem -2^{31} og $2^{31}-1$, ikke bare mellem 0 og 255).

   I denne opgave skal du først lave en `.txt`-fil med 16 tegn i. Lav derefter et Java-program som via fire kald til `readInt()`-metoden fra den udleverede `BitInputStream` læser disse 16 bytes som fire `int`'s og

udskriver hver af dem. [Med lidt viden fra nettet om two's complement repræsentationen (som Java og mange andre sprog bruger) for heltal samt programmet fra forrige opgave, kan man godt checke, om output passer.]

4. Eksamen juni 2013, opgave 7.

5. (∗) Cormen et al. øvelse 15.4-5 (side 397). Hint: lav en (1-dimensional) tabel over $l(i)$ for $i = 1$ til n, hvor $l(i)$ angiver længden af en længste monotont stigende delsekvens *som ender* ved det i'te tal. Dvs. løs dette lidt modificerede problem med dynamisk programmering. Brug til sidst tabellen med løsningerne for dette problem til at løse det oprindelige problem.

   At bruge ovenstående metode er meningen med opgaven. Man kan faktisk også finde længste monotont stigende delsekvens ved at se på inputsekvensen som en streng af tal, og derefter løse et LCS-problem mellem strengen selv og en udgave af den som er blevet sorteret, således at alle tallene kommer i stigende rækkefølge [udfordring: bevis at dette løser det samme problem]. Derefter kan man bruge at vi jo allerede har en metode baseret på dynamisk programmering til at løse LCS. Er køretiden den samme eller er den forskellig for de to måder?

6. (∗) Cormen et al. problem 15-2 (side 405). Hint: For at løse dette direkte med dynamisk programmering, lad delproblemer være givet ved delstrengen $x_i x_{i+1} \ldots x_{j-1} x_j$ for $i \leq j$, og se i analysen på begge ender (dvs. på x_i og x_j) af delproblemet, når man forsøger at relatere det til mindre delproblemer. Herudover ligner analysen lidt den for Longest Common Subsequence.

   Ovenstående er meningen med opgaven. Man kan faktisk også løse dette palindrom-problem ved at løse LCS-problemet for strengen selv og dens omvendte [hård udfordring: bevis dette], hvilket vi jo allerede har en metode baseret på dynamisk programmering for. Er køretiden den samme eller er den forskellig for de to måder?

Studiegrupper

Forslag til fokus for arbejde i studiegrupper (hvis man er i en sådan):

- Forsøg at lave programmeringsopgaverne på forhånd. Diskuter og del

løsningsmetoder i fælleskab, men sørg for at alle skriver deres egen kode.

- Diskuter, hvad der er ens og hvad der adskiller situationer, hvor henholdsvis almindelig divide-and-conquer og dynamisk programmering kan tænkes at være brugbart.

- Forsøg at lave opgaverne med dynamisk programmering på forhånd. Sammenlign svar i studiegruppen. Skiftes til at fremlægge jeres løsning. For de opgaver, hvor alle var gået i stå, forsøg at løse dem igen i fælleskab.