Institut for Matematik og Datalogi
Syddansk Universitet, Odense

10. april 2021
RF

# Opgaver Uge 15

# DM507/DS814/T510040101

## I.A: Løses i løbet af de første øvelsestimer i uge 15

1. Cormen et al. øvelse 15.1-3 (side 370).
2. Cormen et al. øvelse 15.4-1 (side 396).
3. Cormen et al. øvelse 15.4-2 (side 396).
4. Eksamen juni 2010, opgave 4.

## I.B: Løses hjemme inden de næste øvelsestimer i uge 15

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 meget simplere).

## II.A: Løses i løbet af de næste øvelsestimer i uge 15

1. Eksamen juni 2013, opgave 7.

2. 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?]

3. Cormen et al. problem 15-2 (side 405). Hint: For at løse dette direkte med dynamisk programmering, lad delproblemer være givet ved delstrengen $x_i x_{i+1} \dots x_{j-1} x_j$ 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?]

## II.B: Løses hjemme inden øvelsestimerne i uge 16

Nedenstående opgaver er alle opvarmning til projektet del III. Deltagere i T510040101 skal derfor *ikke* regne disse opgaver, og har således ingen lektier for til øvelsestimerne i uge 16.

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.[1]

[1] Et program, der læser en fil, vil normalt *fortolke* dens bits som repræsenterende f.eks. bogstaver, pixels eller lydsvingninger (hvert program fortolker på sin egen måde). Men en fil i sig selv er blot en række bits samlet i grupper af størrelse otte (bytes).

Der er $2^8 = 256$ muligheder for en byte's indhold: 00000000, 00000001, 00000010, 00000011, ..., 11111111. I Java repræsenteres bytes ved `int`'s, hvor man kun bruger værdierne 0 til 255.[2] I Python (Python 3) repræsenteres bytes af byte objects, som man kan tænke på som (immutable) lister af heltal med værdier mellem 0 og 255.[3]

I Java kan `read()`-metoden fra Java-bibliotekets `FileInputStream` læse bytes fra en fil én ad gangen. Hver byte returneres som en `int`. I Python kan kaldet `read(1)` fra file objects læse bytes fra en fil én ad gangen, hvis filen åbnes i binary reading mode (med argumentet `'rb'` i `open`). Hver byte returneres som et byte object af længde én.

I denne opgave skal du lave et Java- eller Python-program som: i) læser en fil én byte ad gangen, ii) undervejs 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
```

I Java skal du bruge `read()`-metoden fra `FileInputStream` fra Java-biblioteket. I Python skal du bruge `read(1)`-metoden fra Pythons file objects. Hver læst byte (et heltal mellem 0 og 255) skal bruges som index i et array (Java) eller liste (Python) af længde 256, hvor plads $i$ fungerer som en tæller for byte nummer $i$. Du skal læse bytes én ad gangen og lave optællingen undervejs.[4]

[2] Der findes en datatype ved navn `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.

[3] Lidt forvirrende vises et helt byte object i Python ved hjælp af ASCII-tegn, hvis det er muligt (f.eks. repræsenteres et byte object `s` med indhold `[120,121,122]` som `b'xyz'` (med `b` for "binary"), jvf. at tegnet `x` i ASCII-tabellen har nummer 120), men enkeltelementer i byte objekter *er* heltal (f.eks. er `s[0]` lig `120`).

[4] Du skal *ikke* starte med at læse hele filen ind og gemme alle dens bytes før du begynder at tælle. Grunden er, at filer kan være store (potentielt større end din computers RAM) og ikke bør opbevares i programmer, hvis det som her nemt kan undgås.

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

2. Opvarming til projektet del III: En byte (en gruppe på otte bits) er den mindste enhed af data, som CPU'er kan håndtere. Derfor er det ikke helt trivielt at tilgå enkelte bits i en fil. På kursets webside er lagt to Java-klasser `BitInputStream` og `BitOutputStream` og et Python library `bitIO.py` med to klasser `BitReader` og `BitWriter`, som giver os mulighed for netop dette. Man behøver *ikke* forstå deres interne virkemåde for at kunne bruge klasserne.[5] Vi vil i denne opgave og den næste træne at bruge de (offentlige) metoder i klasserne.

   I denne opgave skal du i Java lave et 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. Alternativt skal du i Python lave et program, som løser samme opgave ved kald til `readbit()`-metoden fra `BitReader`-klassen i det udleverede library `bitIO.py`.

   Brug en tabel over ASCII-koden for at checke output fra en lille `.txt`-fil med et par få tegn.

3. Opvarming til projektet del III: Udover at kunne læse enkelte bits, kan `BitInputStream` (i Java) og `BitReader` (i Python) også læse fire bytes (32 bits) 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. Alternativt skal du i Python lave et program, som løser samme opgave ved kald til `readint32bits()`-metoden fra `BitReader`-klassen i det udleverede library `bitIO.py`.

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

[5] De virker ved at arbejde en byte forud, og i denne indsætte/læse enkeltbits ved hjælp af shift operationer og bitwise AND og OR.