Institut for Matematik og Datalogi
Syddansk Universitet, Odense

21. april 2022
RF

Opgaver Uge 17

SE4-DMAD

A: Løses i løbet af øvelsestimerne i uge 17

1. Eksamen januar 2007, opgave 2.
2. Eksamen juni 2012, opgave 5.
3. Vi ser her på disjoints sets implementeret via lænkede lister og brug af weighted-union heuristikken for UNION (dvs. at den korteste liste sættes ind i den længste liste). Beskriv hvordan denne implementation kan laves, selv hvis header-objekter ikke indeholder en tail-pointer og ikke gemmer listens længde. Den asymptotiske køretid af operationerne skal selvfølgelig ikke blive ændret. [Hint: løb lister igennem synkront. Husk opdatering af elementers pointer til header.]
4. Cormen et al. øvelse 21.2-2 (side 567). Dine tegninger behøver ikke være lige så detaljerede som i Figur 21.1 (side 565).
5. Cormen et al. øvelse 21.3-2 (side 572). [Hint: løb stien igennem to gange.]
6. Cormen et al. øvelse 21.3-1 (side 572).
7. Cormen et al. øvelse 22.1-1 (side 592).
8. Cormen et al. øvelse 22.1-3 (side 592).
9. Cormen et al. øvelse 22.2-1 (side 601).
10. Cormen et al. øvelse 22.2-2 (side 601).

11. Cormen et al. øvelse 22.2-3 (side 602). NB: Hvis jeres bog ikke er "third printing" (eller senere) af third edition af Cormen et al., står der fejlagtigt "if lines 4 and 14 were removed" - det skal ændres til "if line 18 was removed".

B: Løses hjemme inden øvelsestimerne i uge 18

Opgaverne her er repetition af tidligere stof.

1. Eksamen juni 2011, opgave 3.
2. Eksamen januar 2007, opgave 1. Sidehenvisningerne skal være til side 298 (opgave b) og 294 (opgave c) i vores udgave (tredie) af Cormen et al., i stedet for de angivne sider 261 og 262.
3. (∗) Cormen et al. problem 16.1 (side 446). Erstat spørgsmål **b** med flg. mere generelle: Vis, at hvis der for et møntsæt med møntstørrelser $m_1 = 1, m_2, \ldots, m_k$ gælder at m_i går op i m_{i+1} for alle i, da virker den grådige algoritme fra spørgsmål **a**.

   Hint til spørgsmål **a**: Quarters, dimes, nickels og pennies betyder 25-ører, 10-ører, 5-ører og 1-ører. Du skal vise, at der altid er en optimal løsning bestående af dit første grådige valg samt en optimal løsning til rest-problemet. Det kan hjælpe at se på en optimal løsning, og stille dens mønter op sorteret efter størrelse. Hint til spørgsmål **b**: er ikke så forskellig fra spørgsmål **a**. Hint til spørgsmål **c**: et møntsæt med tre mønter og et beløb n under ti er nok til et modeksempel. Hint til spørgsmål **d**: Man må her bruge dynamisk programmering i stedet for grådighed. Det vil være nok med en tabel $R[i]$ af størrelse $1 \times n$, hvor $R[i]$ indeholder antallet af mønter i en optimal løsning for beløbet i. Tænk derudover lidt som for guldkæde-problemet (se slides om dynamisk programmering)—en optimal løsning for beløb i må indeholde enten en mønt af type 1, eller en af type 2, eller en af type 3, og så videre.

   Bemærk at problem 16.1 viser, at design af et lands møntsæt kræver overvejelser, for at det bliver simpelt (dvs. en naturlig grådig algoritme fungerer) at give penge tilbage.
4. Eksamen juni 2009, opgave 1 a.
5. Eksamen januar 2005, opgave 5.

6. Eksamen juni 2010, opgave 5.

Nedenstående opgaver er alle opvarmning/hjælp til projektet del III. Deltagere i SE4-DMAD behøver derfor ikke lave disse opgaver (men må gerne).

7. 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]

   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
.
.
```

[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).

[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`).

```
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]

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.

8. 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.

9. 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).

[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.

[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.

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.]