

Opgaver Uge 19

DM507/DM578/DS814/SE4-DMAD

I.A: Løses i løbet af de første øvelsestimer i uge 19

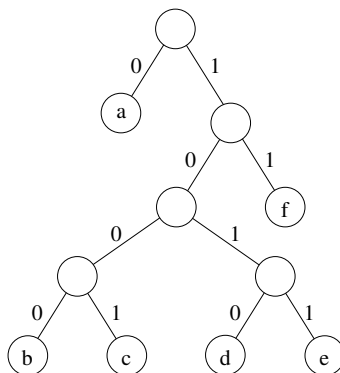
1. Eksamen januar 2007, opgave 2:

Spørgsmål a (8%): Betragt alfabetet med de seks tegn a, b, c, d, e, f. Nedenstående tabel viser, hvor tit hvert enkelt tegn optræder i en given tekst.

a	b	c	d	e	f
33	28	52	20	10	12

Tegn Huffman-træet, som repræsenterer Huffman-koderne for dette eksempel. \square

Spørgsmål b (7%): Brug følgende Huffman-træ



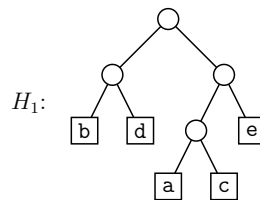
til at dekode 1101001010101011 \square

2. Eksamen juni 2012, opgave 5:

I denne opgave ser vi på en fil som indeholder nedenstående tegn med de angivne hyppigheder.

Tegn	a	b	c	d	e
Hyppighed	100	150	150	250	350

Træet H_1 er et Huffman-træ for denne fil.



Spørgsmål a (5%):

Angiv hvor mange bits filen fylder når den er kodet ved træet H_1 .

□

Spørgsmål b (5%):

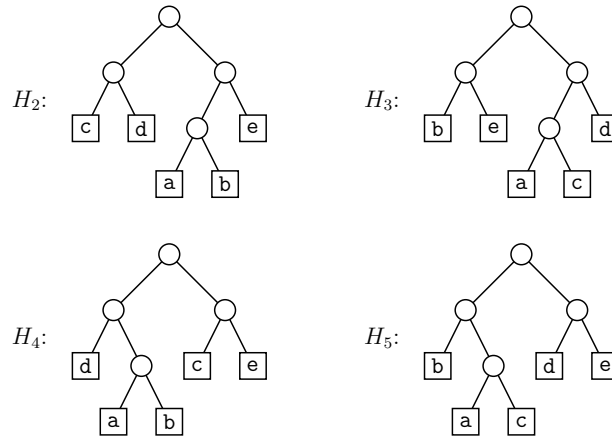
Angiv hvad følgende streng dekodes til ved træet H_1 (under brug af bogens konvention at 0 svarer til venstre og 1 svarer til højre).

1000000110110101

□

Spørgsmål c (5%):

Alle træerne H_2 , H_3 , H_4 og H_5 er optimale for filen med ovenstående tabel. Angiv hvilke af træerne som kan fremkomme ved Huffmans algoritme.



□

3. Cormen et al., 4. udgave, øvelse 19.2-2 (side 526) [Cormen et al., 3. udgave: øvelse 21.2-2 (side 567)]:

Vi ser her på disjoint sets implementeret via lænkede lister og brug af weighted-union heuristikken for UNION (dvs. at den korteste liste appendes til den længste liste). For nedenstående række af operationer, angiv datastrukturens udseende ved afslutningen, samt hvilket element, som de to FIND-SET operationer returnerer. Hvis listen med x_i og listen med x_j har samme længde i en operation $\text{UNION}(x_i, x_j)$, append da listen med x_j til listen med x_i .

```

for  $i = 1$  to 16
    MAKE-SET( $x_i$ )
for  $i = 1$  to 15 by 2
    UNION( $x_i, x_{i+1}$ )
for  $i = 1$  to 13 by 4
    UNION( $x_i, x_{i+2}$ )
UNION( $x_1, x_5$ )
UNION( $x_{11}, x_{13}$ )
UNION( $x_1, x_{10}$ )
FIND-SET( $x_2$ )
FIND-SET( $x_9$ )

```

Dine tegninger behøver ikke være lige så detaljerede som i Figur 19.2 (side 524) [Cormen et al., 3. udgave: Figur 21.2 (side 565)].

4. Vi ser her på disjoint sets implementeret via lænkede lister og brug af weighted-union heuristikken for UNION (dvs. at den korteste liste indsættes 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.]
5. Cormen et al., 4. udgave, øvelse 19.3-1 (side 531) [Cormen et al., 3. udgave: øvelse 21.3-1 (side 572)]:
Vi ser her på disjoint sets implementeret via træer og brug af både union by rank og path compression. Gentag opgave 3 med denne datastruktur. Husk at angive rank for træernes rødder på dine tegninger.
6. Cormen et al., 4. udgave, øvelse 19.3-2 (side 531) [Cormen et al., 3. udgave: øvelse 21.3-2 (side 572)]:
For disjoint sets implementeret via træer: lav en version af FIND-SET, som ikke bruger rekursion. [Hint: løb stien igennem to gange.]

I.B: Løses hjemme inden de næste øvelsestimer i uge 19

Nedenstående opgaver er alle opvarmning/hjælp til projektet del III. Deltagere i DM578 og SE4-DMAD kan springe disse opgaver over (men må også gerne lave dem, naturligvis).

1. Opvarmning 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 Python (Python 3) repræsenteres bytes af byte objects, som man kan tænke på som (immutable) lister af heltal med værdier mellem 0 og 255.²

¹Et 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).

²Lidt 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

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. Hvis man tilgår første (og eneste) element i dette byte objekt, får man et heltal mellem 0 og 255. Hvis byte objektet hedder `b`, tilgås første element som `b[0]`. Når man har nået enden af filen, returnerer `read(1)` et byte object af længde nul i stedet for længde én.

I denne opgave skal du lave et 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
```

Du skal bruge `read(1)`-metoden fra Pythons file objects. Hver læst byte (et heltal mellem 0 og 255) skal bruges som index i en liste 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.³

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 (under punktet Projekt i DM507/DS814) er lagt et Python library `bitIO.py` med to klasser `BitReader` og `BitWriter`, som giver os mulighed for

`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).

³Du 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.

netop dette. Man behøver til projektet *ikke* forstå deres interne virkemåde, man skal blot kunne bruge klasserne.⁴ Vi vil i denne opgave og den næste træne at bruge metoderne i klasserne.

I denne opgave skal du i Python lave et program, som via kald til `readbit()`-metoden fra `BitReader`-klassen i det udleverede library `bitIO.py` læser de enkelte bits i en inputfil 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 lille `.txt`-fil med et par få tegn.

3. Opvarming til projektet del III: Udover at kunne læse enkelte bits, kan `BitReader` 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 Python-program som via fire kald til `readint32bits()`-metoden fra `BitReader`-klassen i det udleverede library `bitIO.py` læser disse 16 bytes som fire `int`'s og udskriver hver af dem.

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

II.A: Løses i løbet af de næste øvelsestimer i uge 19

1. Cormen et al., 4. udgave øvelse 22.1-1 (side 592):

Givet naboliste-strukturen for en graf, hvor lang tid tager det at beregne udgraden (out-degree) for alle knuder? Samme spørgsmålet for indgraden (in-degree). Ændrer disse tider sig, hvis man kun ønsker svarene for én specifik knude?

2. Cormen et al., 4. udgave øvelse 22.1-3 (side 592):

For en orienteret graf G , lad G^T (G transponeret) betegne grafen med alle kanter vendt. Find en algoritme, som konstruerer naboliste-strukturen for G^T ud fra naboliste-strukturen for G . Hvad er køretiden for din algoritme?

⁴Skitse af deres virkemåde: de arbejder en byte forud og læser/indsætter enkeltbits i denne ved hjælp af shift operationer og bitwise AND og OR.

Bemærk: resten af opgaverne er repetition af tidligere stof.

3. Eksamen juni 2011, opgave 3, spørgsmål **a**, **b** og **d** (men ikke **c**):

I denne opgave ser vi på sortering i to special-tilfælde:

1. Alle nøgler er forskellige, og input er omvendt sorteret; d.v.s. nøglerne optræder i faldende orden. Eks: $\{12, 9, 8, 5, 2\}$.
2. Alle nøgler er ens. Eks: $\{5, 5, 5, 5, 5\}$.

I begge tilfælde antages det, at alle nøgler er heltal mellem 0 og n^5 , hvor n er antallet af nøgler, der skal sorteres.

Spørgsmål a (6%): Angiv køretiden for Insertion Sort i hvert af de to tilfælde. ☐

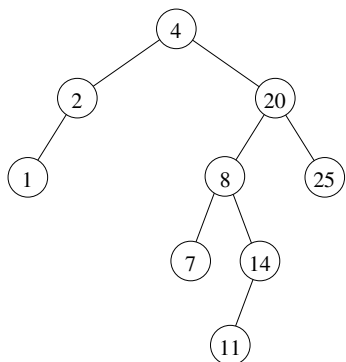
Spørgsmål b (6%): Angiv køretiden for Quicksort i hvert af de to tilfælde. ☐

Spørgsmål c (6%): Angiv køretiden for Heapsort i hvert af de to tilfælde. ☐

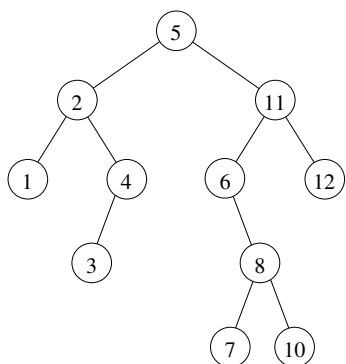
Spørgsmål d (6%): Angiv den bedst mulige køretid for Radix Sort i hvert af de to tilfælde. ☐

4. Eksamen januar 2007, opgave 1 (sidehenvisningerne skal være til side 325 (opgave b) og 321 (opgave c) i Cormen et al., 4. udgave [Cormen et al., 3. udgave: side 298 og 294] i stedet for de angivne sider 261 og 262):

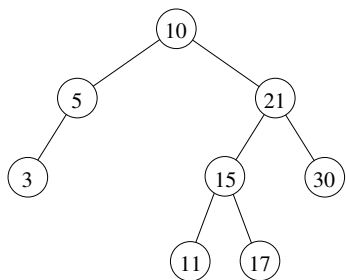
Spørgsmål a (5%): Er nedenstående et binært søgetræ?



Spørgsmål b (8%): Betragt nedenstående binære søgetræ. Tegn træet, som det ser ud, efter at knuden med nøgle 5 er slettet. Du skal bruge algoritmen fra lærebogen (s. 261).



Spørgsmål c (7%): Betragt nedenstående binære søgetræ. Tegn træet, som det ser ud, efter at en knude med nøgle 16 er blevet indsat. Du skal bruge algoritmen fra lærebogen (s. 262).



5. Eksamen juni 2009, opgave 1 a:

Spørgsmål a (7%): Udfør HEAP-EXTRACT-MAX på den binære hob re-præsenteret ved nedenstående array.

10	7	6	5	4	2	3	1	2	3	1	1
----	---	---	---	---	---	---	---	---	---	---	---

Vis hvert skridt. Det kan være en god ide at tegne træ-repræsentationen fremfor array-repræsentationen. \square

6. Eksamen juni 2010, opgave 5:

I denne opgave ønsker vi at udvide binære søgetræer med oplysninger om afstandene mellem de gemte nøgler, og specielt ønsker vi at kunne finde den største afstand i træet mellem en nøgle og dens predecessor.

Mere præcist, hvis et søgetræ gemmer n nøgler $x_1 \leq x_2 \leq x_3 \leq \dots \leq x_n$, er de søgte afstande $(x_2 - x_1), (x_3 - x_2), \dots, (x_n - x_{n-1})$, og vi ønsker at kunne finde den største af disse. Hvis eksempelvis nøglerne gemt i træet er 3, 5, 11, 14, 23 og 30, er afstandene 2, 6, 3, 9 og 7, og største afstand er 9, som opnås mellem nøglen 23 og den predecessor 14.

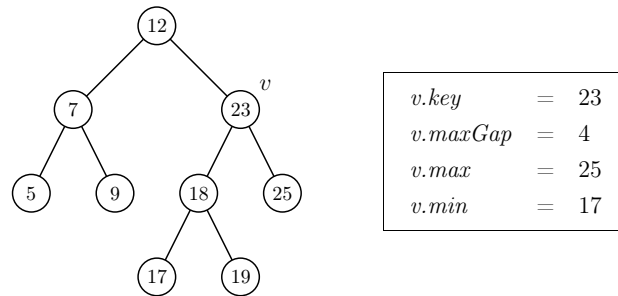
Vi udstyrer nu enhver knude v i søgetræet med følgende tre ekstra informationer (udover den i knuden gemte nøgle $v.key$):

1. Største afstand mellem nøgler gemt i v 's undertræ ($v.maxGap$).
2. Største nøgle gemt i v 's undertræ ($v.max$).
3. Mindste nøgle gemt i v 's undertræ ($v.min$).

(Husk at en knudes undertræ inkluderer knuden selv. Hvis der kun er én nøgle i v 's undertræ, sættes $v.maxGap$ lig nul.)

Specielt kan største afstand i træet herved aflæses af roden r 's information $r.maxGap$ i $O(1)$ tid.

Et eksempel på et binært søgetræ og informationen i en af dets knuder kan ses i Figur 6.



Figur 6: Eksempel på informationen i en knude

Spørgsmål a (6%):

Angiv hvordan en knudes informationer kan bestemmes i $O(1)$ tid ud fra informationerne i knudens to børn, samt knudens og børnenes nøgler.

(Et eller begge af børnene kan være NIL, hvilket giver (simple) specialtilfælde som du ikke behøver beskrive).

Vi lader nu søgetræet være et rød-sort træ.

Spørgsmål b (4%):

Argumentér for at informationerne i træets knuder kan vedligeholdes under indsættelser og sletninger, uden at ændre køretiden $O(\log n)$ for disse.

□

Som sagt kan man i $O(1)$ tid finde den største afstand i træet mellem nøgler og deres predecessors ved at aflæse roden r 's information $r.maxGap$. Vi ønsker nu også at kunne finde en konkret nøgle i træet som har denne afstand til sin predecessor.

Spørgsmål c (5%):

Beskriv en søgeproces som i $O(\log n)$ tid finder en sådan nøgle.

□

II.B: Løses hjemme inden øvelsestimerne i uge 20

1. Eksamen januar 2005, opgave 5:

Denne opgave handler om at sortere n heltal med mange dubletter.

Spørgsmål a: Beskriv en algoritme, der sorterer n heltal i tid $O(n \log(\log n))$, hvis der kun er $O(\log n)$ forskellige tal. Det antages, at to tal kan sammenlignes i konstant tid (d.v.s. i tid $O(1)$).

Hint: Benyt en passende valgt datastruktur til at indsætte i, og saml dubletterne.

Bemærk, at forskellen mellem to tal sagtens kan være $\omega(\log n)$. D.v.s. selvom der kun er $O(\log n)$ forskellige tal, kan der være stor forskel på størrelsen af tallene.

Forklar, hvordan den sorterede følge kan aflæses fra datastrukturen og udskrives i tid $O(n)$.

□

2. (*) Cormen et al., 4. udgave, problem 15.1 (side 446) [Cormen et al., 3. udgave: problem 16.1 (side 446)]:

Denne opgave handler om at finde en måde at betale et heltalsbeløb n med det færrest mulige antal mønter. Overordnet viser opgaven, at design af et lands møntsæt kræver overvejelse, for at det bliver simpelt at give penge tilbage (dvs. at en naturlig grådig algoritme fungerer) i en kontanthandel.

- (a) Vi ser her på det amerikanske møntsæt med quarters, dimes, nickels og pennies (25 cent, 10 cent, 5 cent og 1 cent). Beskriv en grådig algoritme, som finder det færreste antal mønter, der tilsammen udgør et givet beløb på n cent. Bevis, at algoritmen er korrekt.

Hint: Vis, 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 faldende efter størrelse. Argumentet ligner det for næste spørgsmål (som evt. kan løses først). Ud fra dette kan man se den invariant, at algoritmen er ved at bygge en optimal løsning.

- (b) Vis, at hvis der for et møntsæt med mønter af størrelser $m_1 = 1, m_2, \dots, m_k$ gælder, at m_i går op i m_{i+1} for alle i , da virker den grådige algoritme fra spørgsmål a. (Dette spørgsmål er ikke helt det samme som i bogen, men er en generalisering af dette).

Hint: samme som for sidste spørgsmål.

- (c) Find et møntsæt og et beløb n hvor den grådige algoritme ikke virker (dvs. ikke finder det mindste antal mønter). I dit møntsæt skal den mindste mønt have værdien én, hvilket sikrer, at alle beløb n kan opnås.

Hint: et møntsæt med tre mønter og et beløb n under ti er nok.

- (d) Beskriv en algoritme, som i tid $O(kn)$ altid finder det mindste antal mønter til at opnå et beløb n , hvor k er antallet af forskellige mønttyper (hvoraf den mindste har værdien én, således at alle beløb kan opnås).

Hint: Brug 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.