

# Introduktion til algoritmik og datastrukturer

IT-højskolen i København

13. Juni 2000

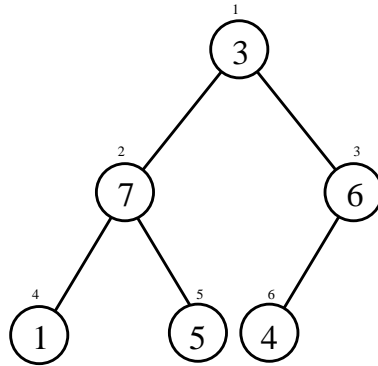
Dette eksamenssæt består af 3 opgaver, hver med 5 delopgaver. Alle opgaverne vægtes ens, og alle delopgaverne i hver opgave vægtes ens. Du har i alt 4 timer til din rådighed. Husk at angive sidetal, navn og studienummer på alle sider i din besvarelse. Eksamenssættet består af 7 nummererede sider.

CLR refererer til "Introduction to Algorithms" af Cormen, Leiserson og Rivest, 18. tryk, 1997.

# Opgave 1

Denne opgave handler om hobe og sortering.

a) Illustrer udførelsen af  $\text{HEAPIFY}(A, 1)$  for hoben  $A$  i figur 1. Anvend stilen fra figur 7.2 i CLR side 143.



Figur 1: Hoben  $A$ , der skal udføres  $\text{HEAPIFY}$  på.

b)  $\text{QUICKSORT}$  (CLR side 154) er defineret vha.  $\text{PARTITION}(A, p, r)$ , som kan bytte rundt på tallene i array'et  $A$  fra indeks  $p$  til  $r$ . Illustrer beregningen af  $\text{PARTITION}(A, 1, 6)$ , hvor

$$A = \langle 7, 8, 9, 4, 5, 6 \rangle$$

I kan benytte Fig. 8.1 side 155 i CLR som en model for illustrationen.

c) Konstruer en sorteringsalgoritme, som har lineær kompleksitet, hvis tallene, der skal sorteres, allerede er *sorterede*; ellers er kompleksiteten  $O(n \lg n)$ . Skriv pseudo-kode.

$\text{QUICKSORT}$  er forventeligt mere effektiv end  $\text{INSERTION-SORT}$  (CLR side 3) for store inddata, men eksperimenter har vist, at  $\text{INSERTION-SORT}$  er mere effektiv end  $\text{QUICKSORT}$ , hvis der skal sorteres få tal.

d) Skriv pseudo-kode for en sorteringsalgoritme, der er en kombination af  $\text{QUICKSORT}$  og  $\text{INSERTION-SORT}$ . Algoritmen skal udnytte, at  $\text{INSERTION-SORT}$  i praksis er mere effektiv end  $\text{QUICKSORT}$ , hvis der skal sorteres færre end  $c$  tal, for en eller anden konstant  $c$ . Det er tilladt at henvise til kode i CLR.

e) Hvad er kompleksiteten af  $\text{HEAPSORT}$ , hvis tallene, der skal sorteres, er *identiske*?

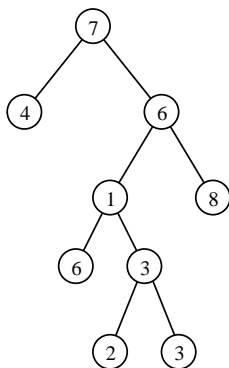
## Opgave 2

Denne opgave handler om binære træer med ikke-negative ( $\geq 0$ ) heltalsvægte i knuderne.

Vi definerer den *vægtede længde* af en vej (simple path) i et træ til at være summen af vægtene i knuderne på vejen, inklusive endeknuder. Dvs. den vægtede længde fra roden i træet i figur 2 til bladet med vægten 2 har en vægtet længde på 19.

Blandt flere veje i et træ er en *vægtet længste vej* en af vejene med den største vægtede længde. F.eks. har den vægtede længste vej (blandt alle mulige veje) i træet i figur 2 værdien 25.

*Vægt-dybden* af et træ er værdien af en vægtet længste vej blandt vejene fra roden til et blad i træet. F.eks. er vægt-dybden 21 for træet i figur 2.



Figur 2: Binært træ med vægte.

**a)** Vis ved hjælp af et lille modeksempel, at den vægtede længste vej i et træ  $T$  ikke nødvendigvis går igennem roden på  $T$ .

I denne opgave betragter vi den sædvanlige repræsentation af et binært træ  $T$ , som brugt i CLR kapitel 13. Dvs. til hver knude  $v$  i træet  $T$  har vi felterne  $left[v]$ ,  $right[v]$ , og  $p[v]$  for venstre barn, højre barn og forældreknuden (parent) henholdsvis. For roden  $r$  er  $p[r] = \text{NIL}$  og for et blad  $w$  er både  $left(w) = \text{NIL}$  og  $right(w) = \text{NIL}$ . Herudover lader vi  $key[v]$  betegne *vægten* i knuden  $v$ . Bemærk, at vi *ikke* antager, at vægtene opfylder *binary-search-property* i denne opgave.

**b)** Lav en procedure  $VÆGTTILROD(v)$ , der beregner den *vægtede længde* af vejen fra knuden  $v$  til roden af  $T$ . Proceduren skal køre i tid  $O(h)$ , hvor  $h$  er højden af  $T$ .

**c)** Lav en rekursiv procedure  $VÆGTDYBDE$ , der beregner vægt-dybden af et binært træ  $T$ .

Vi er nu interesseret i at udvide datastrukturen for det vægtede binære træ  $T$ , så vi hurtigt kan finde vægt-dybden, når vægtene i knuderne kan opdateres løbende. Mere præcist er vi interesseret i at udvide datastrukturen for  $T$ , så vi kan understøtte følgende to operationer:

$\text{OPDATER}(v, x)$ : Ændrer  $T$ , så knuden  $v$  får vægten  $x$ .

$\text{AKTUELVÆGTDYBDE}()$ : Returnerer den aktuelle vægt-dybde for  $T$ .

**d)** Beskriv en udvidelse af datastrukturen for et træ  $T$ , der understøtter de to operationer ovenfor. Køretiden for proceduren  $\text{OPDATER}$  skal være  $O(h)$ , hvor  $h$  er højden af  $T$ , mens  $\text{AKTUELVÆGTDYBDE}$  skal køre i tid  $O(1)$ .  
Vink: Lav et ekstra felt til hver knude  $v$ , der indeholder den aktuelle vægt-dybde af *undertræet* med rod  $v$ .

Vi er nu interesseret i en anden udvidelse af datastrukturen for  $T$ . Først skal træet  $T$  initialiseres, så alle knuder initielt har vægt 1. Derefter skal vi håndtere følgende to operationer:

$\text{SÆTNUL}(v)$ : Ændrer vægten i knuden  $v$  til 0, dvs. sæt  $\text{key}[v]$  til 0.

$\text{NULVEJ}(u, v)$ : Returnerer `true`, hvis og kun hvis den vægtede længde af vejen fra knude  $u$  til knude  $v$  er 0.

**e)** Beskriv en datastruktur for ovennævnte problem. For et træ  $T$  med  $n$  knuder skal løsningen give en *total* køretid på  $O(n \lg^* n)$  for initialisering og udførelse af  $n$   $\text{SÆTNUL}$  og  $\text{NULVEJ}$  operationer.

## Opgave 3

For alle delopgaverne i denne opgave gælder det, at  $A$  er en tabel (array) bestående af  $n$  heltal. Det første tal i tabellen er  $A[0]$ , og det sidste tal i tabellen er  $A[n-1]$ .

I figur 3 er vist en tabel med 8 tal. For denne tabel er  $n = 8$ ,  $A[0] = 15$  og  $A[n-1] = 314$

15	20	31	150	210	214	217	314
----	----	----	-----	-----	-----	-----	-----

Figur 3: En tabel med 8 tal.

**a)** Lad  $A$  være en (ikke nødvendigvis sorteret) tabel med  $n$  tal. Beskriv, hvorledes man i  $O(n \lg n)$  tid kan afgøre, om der er to ens tal i tabellen.

**b)** Lad  $A$  være en (ikke nødvendigvis sorteret) tabel med  $n$  heltal, hvor alle de  $n$  tal har en værdi mellem 1 og  $3n$ . Hvor hurtigt kan de  $n$  heltal sorteres?

**c)** Lad  $A$  være en sorteret tabel med  $n$  tal. Lad  $\text{SUCC}(x)$  være en procedure, der returnerer det mindste tal  $y$  fra  $A$ , som er større end  $x$ . Hvis et sådan tal ikke findes returneres  $\infty$ .  $\text{SUCC}(170)$  på tabellen i figur 3 er således 210. Beskriv, hvorledes  $\text{SUCC}$  kan konstrueres, så den har tidskompleksiteten  $O(\lg n)$ .

Lad  $\text{NEXT}(i)$  være en procedure der givet et indeks  $i$  fra en tabel  $A$  returnere det mindste indeks  $j$ , som er større end  $i$ , hvor tallet  $A[j]$  er lige. Hvis et sådan indeks ikke eksistere returneres  $n-1$ . Betragt tabellen i figur 3. Her vil f.eks.  $\text{NEXT}(0)$  returnere 1,  $\text{NEXT}(1)$  vil returnere 3 og  $\text{NEXT}(5)$  vil returnere 7. Antag, at hvis  $\text{NEXT}(i)$  returnerer  $j$ , så har  $\text{NEXT}$  kompleksiteten  $O(1 + j - i)$ .

Betragt følgende procedure AMOR:

AMOR( $k$ )

1.  $x \leftarrow 0$
2. **for**  $j \leftarrow 1$  **to**  $k$
3.     **do**  $x \leftarrow \text{NEXT}(x)$

**d)** Lad  $A$  være en tabel med  $n$  tal. Hvad returnerer et kald til  $\text{NEXT}(n-1)$ ? Hvad er kompleksiteten for et kald til  $\text{NEXT}(n-1)$ ? Hvad er kompleksiteten for et kald til  $\text{AMOR}(k)$ ? Du skal for hver af følgende kompleksitetsmål  $O(n)$ ,  $O(k)$ ,  $O(n+k)$  og  $O(n * k)$  angive og begrunde om kompleksitetsmålet udtrykker kompleksiteten for  $\text{AMOR}(k)$ .

I den næste delopgave vil vi betragte en tabel  $A$ , hvor alle tallene i tabellen til at starte med er 1. Endvidere antager vi, at tabellen indeholder  $n$  tal, hvor  $n$  kan skrives som  $2^k + 1$  for et heltal  $k \geq 0$  (dvs.  $n$  er et af tallene 2, 3, 5, 9, 17, 33, ...).

I figur 4 er vist en tabel med 9 tal. For denne tabel er  $n = 2^3 + 1 = 9$ ,  $k = 3$ , og  $A[i] = 1$ , for alle indices  $i$ .

1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---

Figur 4: En table med 9 et-taller.

Betragt følgende to procedurer NYETAL og HALVER:

NYETAL()

1.  $i \leftarrow 0$
2. **while**  $i \leq n - 1$
3.     **do**  $A[i] \leftarrow A[i] * 2$
4.          $i \leftarrow i + A[i]$

HALVER( $k$ )

1. **for**  $j \leftarrow 1$  **to**  $k$
2.     **do** NYETAL()

e) Lad  $A$  være en tabel med 9 tal, som alle er 1. Tegn situationen efter et kald til NYETAL. Lad nu  $A$  være en tabel med  $n$  tal, hvor  $n = 2^k + 1$  for et heltal  $k \geq 0$ . Hvad er kompleksiteten af et kald HALVER( $k$ )?