DMA ugeseddel 4

Litteratur

Uge 4 vil vi primært bruge på:

- CLRS introduktion til Part III, side 229–231.
- CLRS kapitel 10.
- CLRS kapitel 6 (ignorér tekst om Master Theorem på side 156).

Mål for ugen

- Overordnet: Kendskab til elementære datastrukturer.
- Kendskab til stakke, køer, hægtede lister, hobe, prioritetskøer.
- Kendskab til heap sort.

Plan for ugen

- Mandag: Stakke, køer og hægtede lister (CLRS kapitel 10).
- Tirsdag: Prioritetskøer og hobe (CLRS kapitel 6).
- Fredag: Heap sort (CLRS kapitel 6). Opsummering og afrunding.

Opgaver til mandag

Pas på: Opgaver markeret med (*) er svære, (**) er meget svære, og (***) har du ikke en chance for at løse. I bunden af ugesedlen finder du flere ekstraopgaver, hvoraf nogle er svære. Dem kan du lave hvis du mangler udfordring eller er færdig med dagens opgaver. Opgaver markeret med (ekstra) kan du evt. gemme til sidst.

1. **Stakke og køer.** Løs følgende opgaver.
   (a) CLRS 10.1-1.
   (b) CLRS 10.1-3.
   (c) CLRS 10.1-2.
2. **Algoritmer på hægtede lister.** Lad L være listen vist i CLRS Figure 10.3 (b), side 237. Løs følgende opgaver.
   (a) Håndkør `Foo(L)`.

(b) Forklar hvad `Foo` gør.

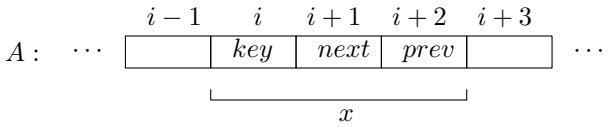
(c) Håndkør `Bar(L.head, 0)`.

(d) Forklar hvad `Bar` gør.

```
Foo(L)
  x = L.head
  c = 0
  while x != NIL
    x = x.next
    c = c + 1
  return c
```

```
Bar(x, s)
  if x == NIL
    return s
  return Bar(x.next, s + x.key)
```

3. **Repræsentation af hægtede lister.** I denne opgave repræsenterer vi hvert element x i en dobbelthægtet liste vha. tre sammenhængende felter i et array A, dvs. x er gemt på felterne $A[i, i+1, i+2]$. På disse tre felter gemmer vi hhv. *key* (nøgleværdien), *next* (adressen på det næste element), og *prev* (adressen på det foregående element):



Adressen på dette listeelement x angives som indekset i, altså indekset på det felt, hvor *key*-værdien er gemt. Dette er ligesom i CLRS Figure 10.6.

(a) Nedenstående array A repræsenterer en dobbelthægtet liste med fem elementer som indeholder *key*-værdierne $7, 2, 11, 5, 8$ i denne rækkefølge, men kun *key*-værdierne er vist i arrayet. Udfyld for hvert element felterne som indeholder adresserne *next* og *prev*.

0	1	2	3	4	5					10				15					20						25	
	2					11							8					7						5		

(b) Vi ønsker at indsætte et nyt listeelement med nøgleværdien 24, som i listens rækkefølge skal være efter 7 og før 2. Vælg et sted i A at lagre elementet, og opdatér *prev*- og *next*-værdierne så A repræsenterer den udvidede liste.

0	1	2	3	4	5					10				15					20						25	
	2					11							8					7						5		

(c) Vi ønsker dernæst at fjerne elementet med nøgleværdien 11 fra listen. Opdatér *prev*- og *next*-værdierne af de øvrige elementer så A repræsenterer den reducerede liste.

0	1	2	3	4	5					10				15					20						25	
	2					11							8					7						5		

4. **Implementation af hægtede lister.** Antag at x er et element i en enkelt-hægtet liste. Løs følgende opgaver.

(a) Antag at x ikke er det sidste element i listen. Hvad er effekten af den følgende kodestump?

```
x.next = x.next.next
```

(b) Lad t være et nyt element der ikke er i listen i forvejen. Hvad er effekten af følgende kodestump?

```
t.next = x.next
x.next = t
```

(c) Hvorfor gør følgende kodestump *ikke* det samme som koden fra ovenstående opgave?

```
x.next = t
t.next = x.next
```

5. **Effektive listeoperationer**

   (a) CLRS 10.2-2.

   (b) CLRS 10.2-3

6. **Sorterede hægtede lister.** Lad L være en enkelt-hægtet liste som består af n elementer, hvor nøgleværdierne er heltal som er gemt i L i sorteret rækkefølge. Løs følgende opgaver.

   (a) Giv en algoritme til at indsætte et nyt element i L således at listen bliver ved med at være sorteret. Din algoritme skal køre i $\Theta(n)$ tid. Skriv pseudokode for din algoritme.

   (b) Professor Gørtz foreslår at man kan forbedre indsættelsesalgoritmen ved at benytte binær søgning. Har hun ret?

7. **Træer.**

   (a) CLRS 10.4-1.

   (b) CLRS 10.4-2.

   (c) CLRS 10.4-3 (ekstra).

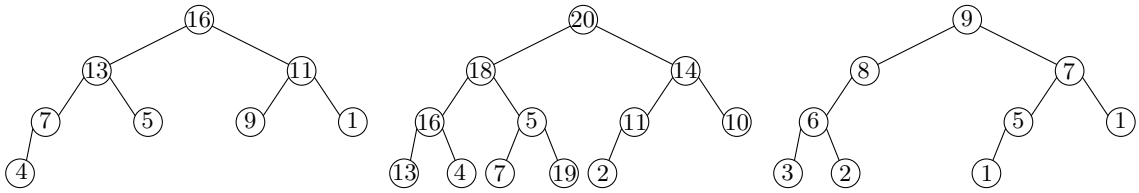
   (d) CLRS 10.4-4 (ekstra).

8. **Sammenligning af lister (ekstra).** Lav CLRS problem 10-1.

Opgaver til tirsdag

1. **Hobeegenskaber og håndkørsel**

   (a) Hvilke af følgende træer er hobe?

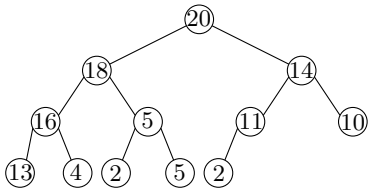


   (b) Hvilke af følgende arrays er hobe?

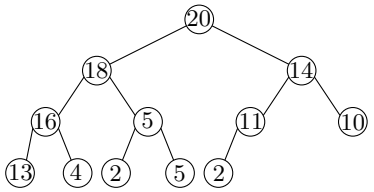
$$A = [9, 7, 8, 3, 4] \qquad B = [12, 4, 7, 1, 2, 10] \qquad C = [5, 7, 8, 3]$$

   (c) Lad $S = (4, 8, 11, 5, 21, *, 2, *)$ være en sekvens af operationer hvor alle tal svarer til en indsættelse af tallet og $*$ svarer til en Extract-Max-operation. Startende med en tom hob H, vis hvordan H ser ud efter hver operation i S.

(d) Tegn hvordan nedenstående max-hob ser ud efter indsættelse af et element med nøgle 19.



(e) Tegn hvordan nedenstående max-hob ser ud efter en EXTRACT-MAX-operation.



2. **Hobe**

   (a) CLRS 6.1-4.

   (b) CLRS 6.1-5.

   (c) CLRS 6.1-6 (ekstra).

   (d) CLRS 6.1-7 (ekstra). *Hint:* Tænk over hvor i tabellen den sidste knude som har børn ligger.

3. **Max-Heapify**

   (a) CLRS 6.2-1.

   (b) CLRS 6.2-3.

   (c) CLRS 6.2-4.

4. **Build-Max-Heap**

   (a) CLRS 6.3-1.

   (b) CLRS 6.3-2.

1 Opgaver til fredag

1. **Heapsort**

   (a) CLRS 6.4-1.

   (b) CLRS 6.4-3. Denne opgave er svær hvis man skal regne køretiden præcist ud (også asymptotisk). Her behøver du bare at undersøge om det tilsyneladende hjælper hvis arrayet A til at begynde med er sorteret eller omvendt sorteret.

2. **Kø med stakke.** CLRS 10.1-6.

3. **Prioritetskøer.**

   (a) CLRS 6.5-1.

   (b) CLRS 6.5-2.

   (c) CLRS 6.5-3.

4. **Listevending**

(a) CLRS 10.2-7.

(b) (*) Lav en rekursiv procedure som vender en liste. Bruger den ligeledes konstant ekstra plads?

5. **0-indekseret hob.** I CLRS er alle hob-funktionerne opskrevet med 1-indeksering. Denne opgave handler om hvilke ændringer der skal laves hvis vi i stedet bruger 0-indeksering, hvor vi gemmer roden af hoben in $A[0]$.

   (a) Opskriv funktionerne PARENT(i), LEFT(i) og RIGHT(i) (CLRS side 152) hvis vi bruger 0-indeksering.

   (b) Hvad skal der ellers ændres i HEAPSORT(A), når man 0-indekserer?

6. **Hob-konstruktion vha. indsættelser.** CLRS problem 6-1.

7. **Rækkefølger som kan laves med en stak.** Lad S være en tom stak. For $i = 1, 2, \ldots, n$ gør vi følgende. Vi putter tallet i på S. Vi popper dernæst 0 eller flere elementer fra S. Efter at tallet n er puttet på S popper vi alle tilbageværende elementer fra S. Vi er interesserede i hvilke rækkerfølger tallene $1, 2, \ldots, n$ kan forlade stakken i. F.eks. kan vi lave rækkefølgen $n, n-1, \ldots, 2, 1$ ved først at poppe efter at n er puttet på, og vi kan lave $1, 2, \ldots, n$ ved at poppe hvert tal lige efter det er puttet på S.

   (a) Opskriv for $n = 4$ to lister: én med alle de rækkefølger man kan lave og én med alle dem man ikke kan lave.

   (b) Hvilke rækkefølger kan laves hvis vi bruger en kø i stedet for en stak?

   (c) (*) Vis at en rækkefølge $p_1, p_2, \ldots, p_n$ ikke kan laves (med en stak) hvis der findes tre indekser i, j, k, hvor $1 \leq i < j < k \leq n$ og $p_i > p_k > p_j$.

   (d) (**) Vis at hvis der *ikke* findes tre indekser i, j, k som beskrevet ovenfor, så kan rækkefølgen godt laves.

   (e) (***) Find en formel for antallet $a(n)$ af rækkefølger der kan laves med et givet n.

Stjerneopgaver (svære ekstraopgaver)

1. **Prioritetspolitik.** Teknokratisk alternativ vil gerne have hjælp til at implementere deres "frisk luft"-politik. Der skal designes et register over borgere og deres indkomst, således man effektivt kan finde dem med lavest indkomst og deportere dem. Specifikt skal systemet understøtte følgende operationer.

   - INDSÆT(c, i): Indsæt person med cpr.-nr. c og årlig indkomst i i systemet.
   - SLET-LAVESTE-INDKOMST(): Fjern og returnér (deportér) person med laveste indkomst.

   (a) Foreslå en effektiv datastruktur M til systemet.

   (b) (*) Antiteknokraterne, et andet ekstrem, får senere magten og indfører et system hvor dem med de højeste indkomster deporteres. Landets statslige IT leverandør laver derfor en ny datastruktur H der understøtter dette system. Senere overtager Fjolleristerne magten. De vil have et system F der kan deportere den person som har den midterste (median) indkomst. Vis hvorledes F kan implementeres vha. H og M.

2. **Nedre grænse for heap sort.** Løs CLRS 6.4-4.

3. **Træer.**

   (a) (*) CRLS 10.4-5.

   (b) (*) CLRS 10.4-6.

4. **Dynamiske mængder med forening.** Vi er interesseret i at vedligeholde en familie af mængder af heltal $\mathcal{F} = \{S_1, \ldots, S_k\}$. Her er hvert S_i en mængde af heltal. Vi vil gerne understøtte følgende operationer.

   - MAKE-SET(x): Tilføj mængden $\{x\}$ til $\mathcal{F}$.
   - REPORT(S_i): Rapportér (f.eks. udskriv til skærmen) alle elementerne i S_i.
   - UNION(S_i, S_j): Tilføj mængden $S_i \cup S_j$ til $\mathcal{F}$. Mængderne S_i og S_j slettes fra $\mathcal{F}$.
   - DISJOINT-UNION(S_i, S_j): Ligesom UNION(S_i, S_j) på nær at det antages at S_i og S_j er disjunkte, dvs., at S_i og S_j ikke har nogle elementer til fælles. Hvis S_i og S_j ikke er disjunkte er resultatet af operationen udefineret.

   Eksempel: Lad $\mathcal{F}$ bestå af 3 mængder $S_1 = \{2, 12, 5, 13\}$, $S_2 = \{6, 7, 1\}$ og $S_3 = \{8, 1, 7\}$. Et kald til DISJOINT-UNION(S_1, S_2) producerer mængden $S_1 \cup S_2 = \{2, 12, 5, 13, 6, 7, 1\}$, hvorefter $\mathcal{F}$ består af $S_1 \cup S_2$ og S_3. Et kald til UNION($S_1 \cup S_2, S_3$) producerer mængden $S_1 \cup S_2 \cup S_3 = \{2, 12, 5, 13, 6, 7, 1, 8\}$ hvorefter $\mathcal{F}$ består af $S_1 \cup S_2 \cup S_3$. Løs følgende opgaver.

   (a) Giv en datastruktur, der understøtter MAKE-SET og DISJOINT-UNION i $O(1)$ tid og REPORT(S_i) i $\Theta(|S_i|)$ tid. *Hint:* Benyt en passende listedatastruktur.

   (b) Udvid din datastruktur således at den også understøtter UNION og analysér tidskompleksiteten af din løsning.

   (c) (*) Giv en datastruktur, der understøtter MAKE-SET i $O(1)$ tid, REPORT(S_i) i $\Theta(|S_i|)$ tid og UNION(S_i, S_j) i $\Theta(|S_i| + |S_j|)$ tid (bemærk at DISJOINT-UNION ikke skal understøttes).

5. (*) Vis at INSERT, EXTRACT-MAX og INCREASE-KEY bibeholder hobeordenen.

6. (*) CLRS 6.5-9.

7. **Hobegenskaber (*)**

   (a) Lad T være et komplet binært træ af højde h. Højden h er defineret som antallet af kanter på vejen fra et blad til roden. Vis at antallet af knuder i T er $n = 2^{h+1} - 1$. Hint: vi har at $n = 1 + 2 + 4 + \ldots + 2^h$. Multiplicér summen med 2 og træk summen fra. Eller tænk på summen i binær repræsentation.

   (b) Lad $n' = 2^{h+1}$ for et heltal $h \geq 1$. Definér $S = n'/4 \cdot 1 + n'/8 \cdot 2 + n'/16 \cdot 3 + n'/32 \cdot 4 + \ldots + n'/2^{h+1} \cdot h$, og vis at $S = \Theta(n')$. (Hint: Udregn $S - S/2$.) Udled af denne udregning at BUILD-MAX-HEAP på en tabel af størrelse $n = 2^{h+1} - 1$ (svarende til et komplet binært træ af højde h) tager $\Theta(n)$ tid.

   (c) CLRS 6.1-1

   (d) CLRS 6.1-2

   (e) CLRS 6.3-3

8. **Prioritetskøoperationer.** Vi vil gerne tilføje nogle operationer til vores (maks)prioritetskø. Vi er interesseret i at tilføje følgende operationer. I nedestående er x og y objekter. Værdierne $x.key$ og $y.key$ er deres nøgler i prioritetskøen (objekterne kan have andre satellitdata). Hvis en implementation af prioritetskøen sker som i CLRS kapitel 6 kan tabellen være en tabel af objekter – hvert objekt kan altså indeholde mere information end blot nøgleværdien. Fx kan x og y tænkes at have en attribut i, således at $x.i$ og $y.i$ angiver deres plads (index) i en tabel som definerer prioritetskøen.

   - REMOVE-LARGEST(m): fjern de m største elementer i hoben.
   - DELETE(x): fjern elementet x fra prioritetskøen.
   - FUSION(x, y): fjern x og y fra hoben og tilføj elementet z med nøgle $z.key = x.key + y.key$.

- FIND-LARGE(k): returner de elementer i hoben med nøgle $\geq k$.
- EXTRACT-MIN(): fjern og returnér et element med den mindste nøgle.

Vi vil gerne implementere disse operationer, mens vi stadig bibeholder kompleksiteten af de sædvanlige prioritetskøoperationer. Lad n være antallet af elementer i prioritetskøen. Løs følgende opgaver.

(a) Udvid prioritetskøen til at understøtte REMOVE-LARGEST(m) i $O(m \log n)$ tid.

(b) Udvid prioritetskøen til at understøtte DELETE og FUSION i $O(\log n)$ tid.

(c) (*) Udvid prioritetskøen til at understøtte FIND-LARGE(k) i $O(m)$ tid, hvor m er antallet af elementer med nøgle $\geq k$.

(d) (**) Udvid prioritetskøen til også at understøtte EXTRACT-MIN i $O(\log n)$ tid.

Bemærkninger: Nogle opgaver er stærkt inspireret af opgaver stillet af Philip Bille og Inge Li Gørtz i kurset Algoritmer og Datastrukturer på DTU,
`http://www2.compute.dtu.dk/courses/02105+02326/2015/#generelinfo`.