DMFS 2023: Ugeseddel for Mikkel Abrahamsen's Lecture on Heaps

Litteratur

CLRS kapitel 6.

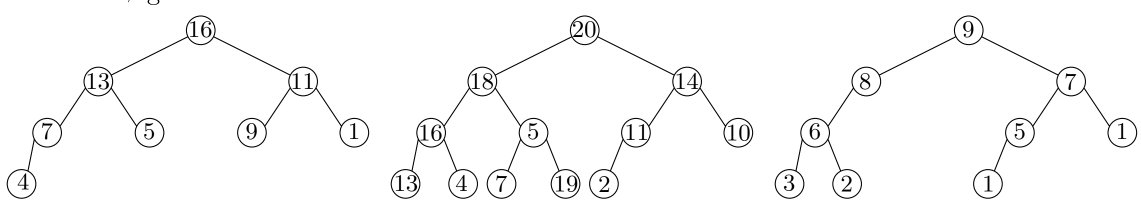
Mål

Kendskab til heap sort.

Opgaver

1. **Hobeegenskaber og håndkørsel**

   (a) Hvilke af følgende træer er hobe?

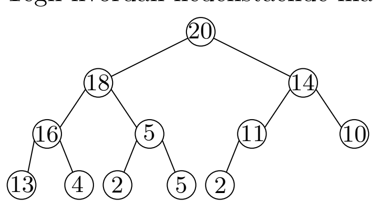
   

   (b) Hvilke af følgende arrays er hobe?

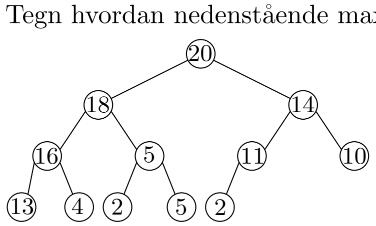
   $$A = [9, 7, 8, 3, 4] \qquad B = [12, 4, 7, 1, 2, 10] \qquad C = [5, 7, 8, 3]$$

   (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 6.1-6.
   (c) CLRS 6.1-7 (ekstra).
   (d) CLRS 6.1-8 (ekstra). *Hint:* Tænk over hvor i tabellen den sidste knude som har børn ligger.

3. **Max-Heapify**
   (a) CLRS 6.2-1.
   (b) CLRS 6.2-4.
   (c) CLRS 6.2-5.

4. **Build-Max-Heap**
   (a) CLRS 6.3-1.
   (b) CLRS 6.3-3.

5. **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.

6. **Prioritetskøer.**
   (a) CLRS 6.5-1.
   (b) CLRS 6.5-2.
   (c) CLRS 6.5-3.

7. **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 $\textsc{Parent}(i)$, $\textsc{Left}(i)$ og $\textsc{Right}(i)$ (CLRS side 162) hvis vi bruger 0-indeksering.
   (b) Hvad skal der ellers ændres i $\textsc{Heapsort}(A)$, når man 0-indekserer?

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

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.
   - $\textsc{Indsæt}(c, i)$: Indsæt person med cpr.-nr. c og årlig indkomst i i systemet.
   - $\textsc{Slet-Laveste-Indkomst}()$: 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. **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).

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

5. (*) CLRS 6.5-11.

6. **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 6.3-4.

7. **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.

8. **Delsummer.** Lad $A[0, \ldots, n-1]$ være en tabel af heltal. Vi er interesseret i følgende operationer på A.

   - Sum(i, j): beregn $A[i] + A[i+1] + \cdots + A[j]$.
   - Change(i, x): sæt $A[i] = x$.

   Løs følgende opgaver. (Det er i den sidste delopgave at man muligvis har brug for en hobelignende struktur.)

   (a) Giv en datastruktur, der understøtter Sum i $O(1)$ tid og bruger $O(n^2)$ plads.
   (b) (*) Giv en datastruktur, der understøtter Sum i $O(1)$ tid og bruger $O(n)$ plads.
   (c) (***) Giv en datastruktur, der understøtter både Sum og Change i $O(\log n)$ tid og bruger $O(n)$ plads.

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`.