Eksaminatorie-timer

DM507: Algoritmer og datastrukturer

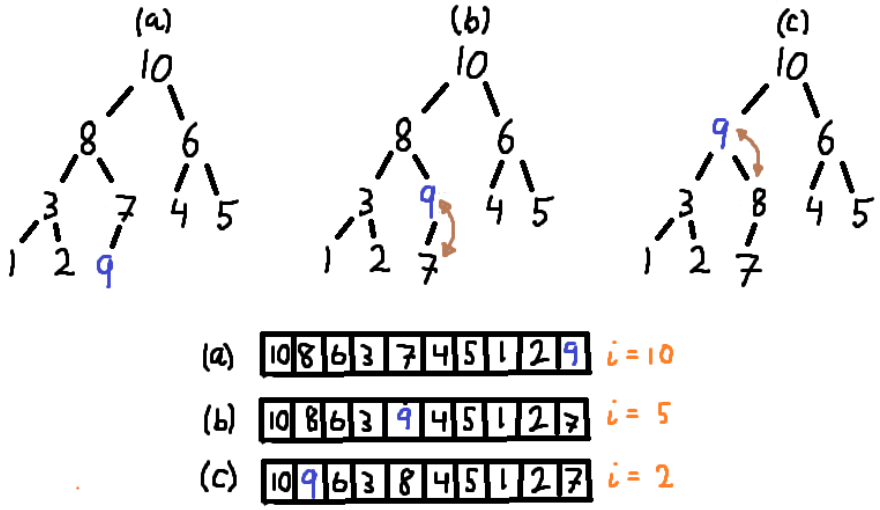
Løsninger

Uge 8 # 2 2021

Del A

Opgave 1 - Eksamen januar 2008, opgave 1 b

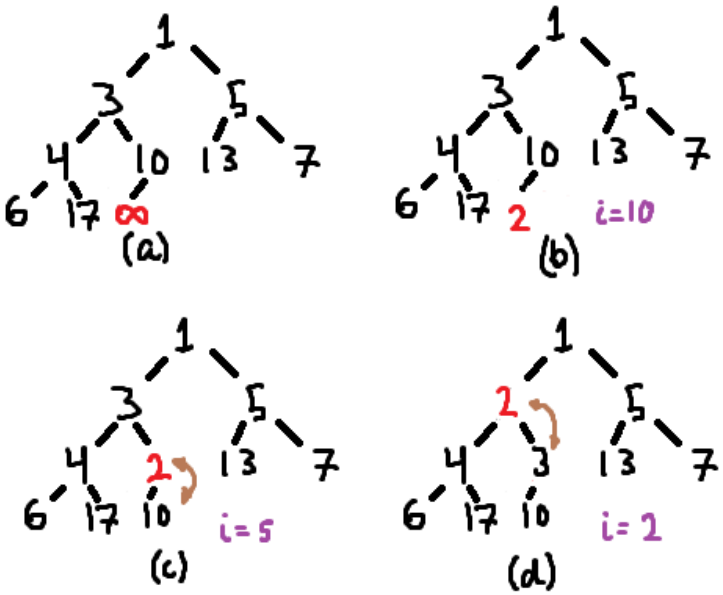
Indsæt nøglen 9 i den binære binary max-heap repræsenteret ved arrayet $\langle 10, 8, 6, 3, 7, 4, 5, 1, 2 \rangle$. Vis resultatet før hvert gennemløb af `while`-løkken i algoritmen på side 164 (3. udgave Cormen et al.). Du må gerne tegne det som en træstruktur i stedet for et array.



Essensen: Indsæt ny nøgle som det sidste element i array, og gentag byt den med dens forældre indtil forældren har en nøgle større end (eller lig) den nye nøgle.

Opgave 2 - Eksamen januar 2006, opgave 1 b

Betragt følgende binære hob (binary heap) $\langle 1, 3, 5, 4, 10, 13, 7, 6, 17 \rangle$. Nu indsættes et element med prioitet 2. Tegn hoben, som den ser ud efter denne indsættelse.



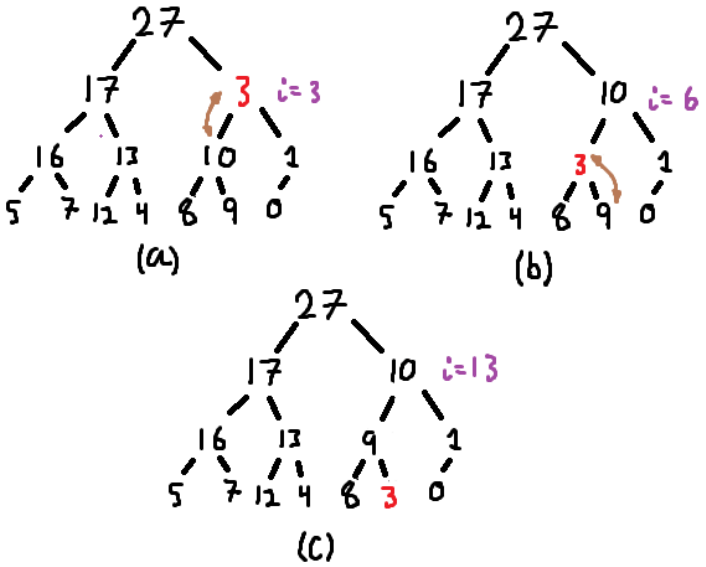
Essensen: Indsæt ny nøgle som det sidste element i array, og gentag byt den med dens forældre indtil forældren har en nøgle mindre end (eller lig) den nye nøgle.

Opgave 3 - Cormen et al. øvelse 6.2-1

Illustrer `MAX-HEAPIFY(A, 3)`, ligesom Figure 6.2 (fra Cormen et al.), på arrayet

$$A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$$

.



Essensen: Givet en knude med to undertræer, som hver især overholder heap-orden, få hele knudens træ til at overholde heap-orden. Dette gøres rekursivt ved at bytte nøgle med barnet med den største nøgle, og derefter køre `Max-Heapify` på dette barn.

Opgave 4 - fra ugeseddel

Udfør `Heap-Extract-MaX(A)` på nedenstående max-heap A.

A:	21	18	10	12	8	9	4	7	5	2

Følgende sker:

- A er ikke tom.
- Udfør `x = A[1] = 21`. Dvs. gem max-elementet i variable.
- Udfør `A[1] = A[A.heap-size]`. Dvs. flyt sidste element op som første.
- Gør array/heap en mindre.
- Udfør `Max-Heapify(A,1)`. Dvs. genopret heap-orden rekursivt. Følgende opdateringer sker:

A':	2	18	10	12	8	9	4	7	5

A":	18	2	10	12	8	9	4	7	5

A"':	18	12	10	2	8	9	4	7	5

A"":	18	12	10	7	8	9	4	2	5

Opgave 5 - implementer Quicksort

Se koden i `quick_sort/`. For den almindelige udgaven se `QuickTiming.java`. For median-pivot udgaven se `QuickMedianTiming.java`. For Java udgaven se `JavaSortTiming.java`. Sammenlignes Mergesort (fra tidligere opgave) og Quicksort, så burde man se Quicksort være hurtigere. Sammenlignes median-pivot udgaven og den almindelige, så ser man nok median-pivot er en smule hurtigere. Sammenlinger man Java's implementation, så vil man højst sandsynligt se den er hurtigere.

Del B

Opgave 1 - Cormen et al. øvelse 6.5-9

Giv en $O(n \lg k)$ algoritme til at merge k sorterede lister sammen til én sorteret liste, hvor n er det totale antal elementer i alle input listerne (Hint: brug en min-heap for k-vejs merging).

Lav n gennemløb, hvor det mindste element findes vha. `EXTRACT-MIN` fra en min-heap, der indeholder det mindste/første element fra hver af de k lister. Efter `EXTRACT-MIN` opdateres min-heapen med elementet, som kommer efter det udtagne element i den givne liste.

- Korrekthed: Klart korrekt! Da de k lister er sorterede, så er det mindste element det første element i en af listerne. En min-heap er bare en effektive måde at finde det mindste element på.
- Køretid: man laver n gennemløb, hvor man udfører én `EXTRACT-MIN` og (måske[1]) én `MIN-HEAP-INSERT` som hver tager $O(\lg k)$ tid; altså en $O(n \lg k)$ worst-case køretid.

Note: En forbedring ville nok være at skrive `EXTRACT-MIN` og `MIN-HEAP-INSERT` sammen i dette tilfælde – vil højst sandsynligt give mindre konstanter.

Opgave 2 - fra opgaveark

Bevis at de beregnede indexer i `Parent`, `Left` og `Right` på side 152 er korrekte (dvs. for en knude på index i opnås index af hhv. dens forælder, venstre barn og højre barn).

- `Left` og `Right`: Vha. simple induktion bevises det, at det venstre og højre barn af elementet på indeks k er hhv. $2k$ og $2k + 1$.

  Basis: For roden med indeks 1 gælder

  $$\texttt{Left(1)} = 2 \cdot 1 = 2 \quad \text{(Sand)}$$
  $$\texttt{Right(1)} = 2 \cdot 1 = 3 \quad \text{(Sand)}$$

  Induktionsantagelse: Antag indeks af det venstre og højre barn for indeks k er hvv. $2k$ og $2k + 1$.

  Induktionsskridt ($k \geq 1$): Ud fra binære træ illustreret på Figur 1, så følger det af induktionsantaglesen, at det venstre og højre barn af indeks $k + 1$ er $2k + 2$ og $2k + 3$ ((*)). Da

  $$\texttt{Left(k+1)} = 2k + 2 = 2(k + 1)$$
  $$\texttt{Right(k+1)} = 2k + 3 = 2(k + 1) + 1$$

  konkluderer vi `Left` og `Right` beregner de korrekte indekser.

  □

[1] Hvis én af de k lister tømmes, så behøves min-heapen ikke opdateres - det er dog en implementations detalje.

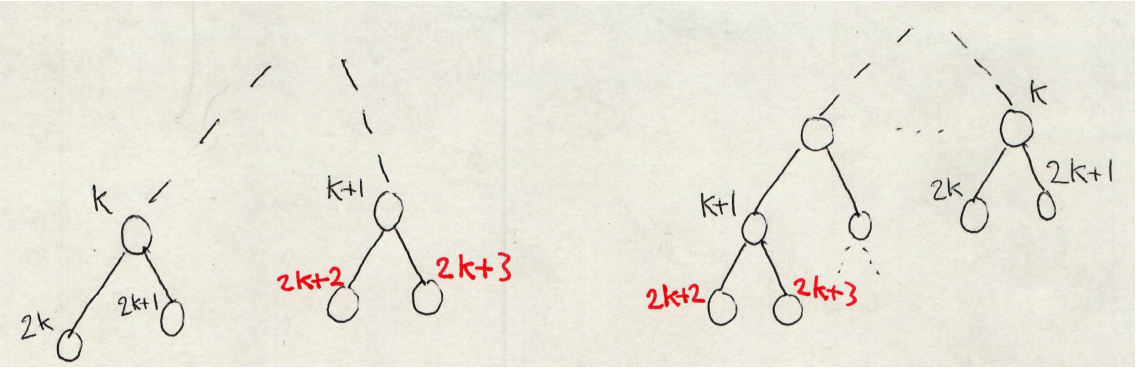


Figure 1: Relative indekser mellem børn og forældre i et binært træ med heap facon.

(*): Man kan se det som to processer, der løber igennem træet, og de er forskudt et lag. Hver gang den første process passerer én knude, så passerer den anden process to knuder – nemlig de to børn af den knude den første process passerede. Dette gælder også når processerne hopper et lag ned (den næste knude er den første i det næste lag - se højre del af Figur 1).

- `Parent`: For en forældre knude med indeks i gælder, at den venstre og højre barn er på hhv. indeks $2i$ og indeks $2i+1$. Da

$$\texttt{Parent(2i)} = \lfloor \frac{2i}{2} \rfloor = \lfloor i \rfloor = i$$
$$\texttt{Parent(2i+1)} = \lfloor \frac{2i+1}{2} \rfloor = \lfloor i + \frac{1}{2} \rfloor = i$$

konkluderer vi `Parent` beregner de korrekte indekser. □

Opgave 3 - Cormen et al. problem 6.2

En *d-ary heap* er ligesom en binær heap, men indre knuder (ikke blade) kan have d børn i stedet for 2 børn.

a) Hvordan kan man repræsentere en d-ary heap i et array?

Udlæg d-ary heap i array ligesom binær heap. Betragt nedenstående figur af træstrukturen. Der bruges 0-indeksering for knuder (fremfor 1-indeksering som med binær heap). Observér, indekserne for knuder på venstre-stien er 0, $0+1$, $0+1+d$ osv. Derudover tælles børnene fra 1 (fremfor 0).

Navigation: Benyt i stedet følgende navigationsformler:

- Voksen til barn: Den første knude i lag k har indeks $i = 1 + d + d^2 + \cdots + d^{k-1}$. Den sidste knude i lag k har indeks $i = d + d^2 + \cdots + d^k$. Heraf følger

$$d + d^2 + \cdots + d^k = (\underbrace{1 + d + d^2 + \cdots + d^{k-1}}_{\text{indeks } i}) \cdot d = i \cdot d$$

  Knuden der kommer efter den sidste knude i lag k er den første knude i lag $k+1$. Derfor er den første knude i lag $k+1$, som er det første barn af den første knude i lag k, givet ved indeks $i \cdot d + 1$. Mere generelt, for barn t, $1 \leq t \leq d$ gælder formlen $i \cdot d + t$. Gælder denne formel kun for knuder på venstrestien? Nej – når vi går l knuder ind i et lag til indeks $i' = i + l$, så går vi $d \cdot l$ knuder ind i det efterfølgende lag. For knude $i' = i + l$ er det t'te barn

$$i \cdot d + d \cdot l + t = (i + l) \cdot d + t = i' \cdot d + t$$

  .

- Barn til voksen: For et barn på indeks j gælder, at `parent(j)` $= \lfloor \frac{j-1}{d} \rfloor$. Bevis: et barn kan skrives på formen $i \cdot d + t$, hvor i er indeks for forældren.

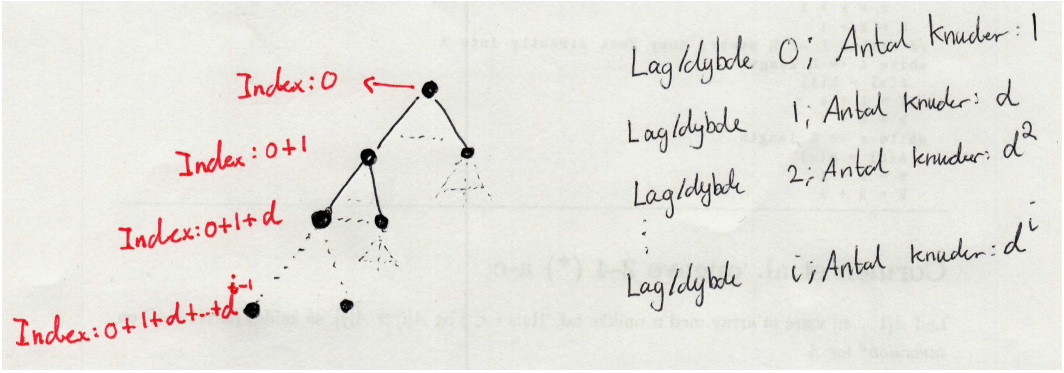
  Af `parent(i · d + t)` følger

$$\begin{aligned}
\left\lfloor \frac{(i \cdot d + t) - 1}{d} \right\rfloor &= \left\lfloor \frac{i \cdot d + (t-1)}{d} \right\rfloor \\
&= \left\lfloor \frac{i \cdot d}{d} + \frac{t-1}{d} \right\rfloor \\
&= \left\lfloor i + \frac{t-1}{d} \right\rfloor \\
&= \left\lfloor i \right\rfloor \quad , \text{ hvor } \frac{t-1}{d} < 1 \\
&= i
\end{aligned}$$

b) Hvad er højden af en d-ary heap med n element som funktion af n og d?

Samme idé som [1, p.22]. Observér, at for et træ med n knuder af højde h gælder

$$n > \underbrace{1 + d + d^2 + \cdots + d^{h-1}}_{\#\text{ knuder fuldt træ af højde h-1}} = \sum_{i=0}^{h-1} d^i = \frac{d^h - 1}{d - 1}$$



Heraf følger

$$\begin{aligned} n > \frac{d^h - 1}{d - 1} \Leftrightarrow \\ n \cdot (d-1) > d^h - 1 \Leftrightarrow \\ n \cdot (d-1) + 1 > d^h \Leftrightarrow \\ h < \log_d(n \cdot (d-1) + 1) \end{aligned}$$

Eftersom

$$\begin{aligned} h &< \log_d(n \cdot (d-1) + 1) \\ &= \log_d(nd - (n-1)) \\ &\leq \log_d(nd) \\ &= \log_d(n) + \log_d(d) \\ &= \log_d(n) + 1 \end{aligned}$$

følger det at højden er $O(\log_d n)$.

c) Giv en effektiv implementation af `EXTRACT-MAX` i en d-ary heap. Analyser køretiden som funktion af d og n.

- `HEAP-EXTRACT-MAX` fra Cormen et al. kan stort set genbruges. Man skal "bare" lave den brugte `MAX-HEAPIFY` om, så man i stedet betragter en knude med d undertræer fremfor 2 undertræer.
- Køretid: I værste fald skal man bevæge sig fra roden ned til et blad, og ved hvert kald til `MAX-HEAPIFY'` skal de d børn tjekkes. Altså, så er worst-case køretiden $O(d \cdot \log_d n)$.

d) Giv en effektiv implementation af `INSERT` i en d-ary heap. Analyser køretiden som funktion af d og n.

- `MAX-HEAP-INSERT` fra Cormen et al. kan stort set genbruges - den skal fra kalde `INCREASE-KEY(A,A.heap_size,k)` som beskrevet i (e).
- Køretid: Pga. den kalder `INCREASE-KEY` med worst-case køretid $O(h) = O(\log_d n)$, så er worst-case køretiden for `INSERT` også $O(\log_d n)$.

e) Giv en effektiv implementation af `INCREASE-KEY(A,i,k)` i en d-ary heap, som giver en fejl hvis $k < A[i]$, men ellers sætter $A[i] = k$ og så opdaterer den d-ary max-heap passende. Analyser køretiden som funktion af d og n.

- `HEAP-INCREASE-KEY` fra Cormen et al. kan stort set genbruges - man skal bare bruge `Parent` metoden for d-ary heaps istedet.
- Køretid: I værste fald skal man bevære sig fra de dybeste lag (et blad) til roden, så worst-case køretiden bliver $O(h) = O(\log_d n)$.

References

[1] Rolf Fagerberg. Sortering. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F21/sortingSlides.pdf`, 2021.