Föreläsning
Datastrukturer (DAT036)

Nils Anders Danielsson

2012-11-07

Idag

- Prioritetsköer.
- Binära heapar.
- Binomialheapar.

Prioritetsköer

Köer där varje element har viss prioritet.

Gränssnitt (exempel):

- Konstruerare för tom kö.
- `insert`: Lägger till element.
- `delete-min`:
  Tar bort och ger tillbaka *minsta* elementet.
- `find-min`: Ger tillbaka minsta elementet.
- `decrease-key`: Ändrar ett elements prioritet.
- `merge`: Slår ihop två köer.

Prioritetsköer

Några tillämpningar:

- Schemaläggning av processer.
- Sortering.
- Dijkstras algoritm (labb 3).

Labb 2: Implementera prioritetskö.

Övning

Implementera prioritetskö-ADTn
(`insert` + `delete-min`) med listor.
Analysera tidskomplexiteten.

Binära heapar

Kompletta binära träd med
heapordningsegenskapen.

Heapordningsegenskapen

Varje nod är mindre än eller lika med alla sina barn.

Komplett binärt träd

Så lågt som möjligt, alla nivåer helt fyllda
utom möjligtvis den sista, som är fylld från vänster.

Kompletta binära träd

Perfekt binärt träd

Så lågt som möjligt, alla nivåer helt fyllda.

Storlek (som funktion av höjden):

$$
\begin{aligned}
S(-1) &= 0 \\
S(0) &= 1 \\
S(1+h) &= 1 + 2S(h), \quad h \geq 0
\end{aligned}
$$

Kompletta binära träd

Perfekt binärt träd

Så lågt som möjligt, alla nivåer helt fyllda.

Storlek (som funktion av höjden):

$$
\begin{aligned}
S(-1) &= 0 \\
S(0) &= 1 \\
S(1+h) &= 1 + 2S(h), \quad h \geq 0
\end{aligned}
$$

$$
S(h) = 2^{1+h} - 1
$$

Kompletta binära träd

Storlek av komplett binärt träd med höjd $h \geq 0$:

$$2^h \leq S(h) \leq 2^{1+h} - 1$$

Höjd av komplett binärt träd med storlek $n \geq 1$:

$$2^{H(n)} \leq n \leq 2^{1+H(n)} - 1$$
$$\log_2(1+n) - 1 \leq H(n) \leq \log_2 n$$

Slutsats: $S(h) = \Theta(2^h)$, $H(n) = \Theta(\log n)$.

Implementation av binära heapar

- Tom kö: Tomt träd.
- `find-min`: Ge tillbaka roten.
- `insert`: Stoppa in sist. Bubbla upp.
- `delete-min`: Ta bort roten.
  Stoppa in sista elementet överst. Bubbla ned.
  Ge tillbaka gamla roten.

Bubbla upp/ned tills heapordningsegenskapen återställts.

Implementation av binära heapar

- Tom kö: Tomt träd.
- `find-min`: Ge tillbaka roten.
- `insert`: Stoppa in sist. Bubbla upp.
- `delete-min`: Ta bort roten.
  Stoppa in sista elementet överst. Bubbla ned.
  Ge tillbaka gamla roten.

Bubbla upp/ned tills heapordningsegenskapen återställts.

Heapanimation.

Tidskomplexitet

Om man kan hitta noderna snabbt:

- Tom kö: $O(1)$.
- `find-min`: $O(1)$.
- `insert`: $O(\log n)$.
- `delete-min`: $O(\log n)$.

(Givet att jämförelser tar konstant tid.)

De flesta noderna är långt ned i trädet.
Medelkomplexitet för `insert`: $O(1)$.

Implementation av binära heapar

Kan representera trädet med array (labb 2).

- Rot på position 1.
- Nod is vänstra barn: $2i$.
- Nod is högra barn: $2i + 1$.
- Nod is förälder ($i > 1$): $\lfloor i/2 \rfloor$.

(Varning för cachemissar.)

Övning

Implementera `decrease-key`.
Antag att ni känner till elementets position i trädet.

Övning

Implementera `delete`.
Antag att ni känner till elementets position i trädet.

Övning

Implementera `decrease-key`.
Antag att ni känner till elementets position i trädet.

Ändra prioritet, bubbla åt rätt håll.

Övning

Implementera `delete`.
Antag att ni känner till elementets position i trädet.

Övning

Implementera `decrease-key`.
Antag att ni känner till elementets position i trädet.

Ändra prioritet, bubbla åt rätt håll.

Övning

Implementera `delete`.
Antag att ni känner till elementets position i trädet.

Ändra prioriteten till $-\infty$, kör `delete-min`.

Känner till positionen?

Kan använda extra datastruktur för att hitta nod snabbt.

Exempel: Hashtabell.

`build-heap`

`build-heap`: Konverterar lista e d till heap.

- Stoppa in alla elementen i godtycklig ordning.
- Bubbla *ned* ett element i taget,
  med början på det nedersta, högraste.
  (Kan skippa alla löv.)

Heapordningsegenskapen uppfylls
(kan bevisas med induktion).

`build-heap`: tidskomplexitet

- Tid för viss nod: $O(\text{nodens höjd})$.
  (Givet $O(1)$-jämförelser.)
- Total tid: $O(\text{summan av alla höjder})$.
- Nästan alla noder är långt ned.

`build-heap`: tidskomplexitet

Antalet noder på nivå i:

$$L(0) = 1$$
$$L(1+i) \leq 2L(i)$$

$$L(i) \leq 2^i$$

`build-heap`: tidskomplexitet

Tidskomplexitet:

$$
\begin{aligned}
T(n) &\leq \sum_{i=0}^{H(n)} L(i)(H(n) - i) \\
&\leq \sum_{i=0}^{H(n)} 2^i (H(n) - i) \\
&\stackrel{\text{def}}{=} S
\end{aligned}
$$

`build-heap`: tidskomplexitet

$$S = \sum_{i=0}^{H(n)} 2^i (H(n) - i)$$

$$\begin{aligned} S &= 2S - S \\ &= \sum_{i=0}^{H(n)} 2^{1+i} (H(n) - (1+i) + 1) - S \\ &= \sum_{j=1}^{1+H(n)} (2^j (H(n) - j) + 2^j) - S \end{aligned}$$

build-heap: tidskomplexitet

$$S = \sum_{i=0}^{H(n)} 2^i (H(n) - i)$$

$$S = \sum_{j=1}^{1+H(n)} (2^j (H(n) - j) + 2^j) - S$$

$$= 2^{1+H(n)} (H(n) - (1 + H(n))) - H(n) + \sum_{j=1}^{1+H(n)} 2^j$$

`build-heap`: tidskomplexitet

$$
\begin{aligned}
&= 2^{1+H(n)}(H(n) - (1 + H(n))) - H(n) + \sum_{j=1}^{1+H(n)} 2^j \\
&= -2^{1+H(n)} - H(n) + 2(2^{1+H(n)} - 1)/(2 - 1) \\
&= 2^{1+H(n)} - H(n) - 2 \\
&= O(n)
\end{aligned}
$$

Dessutom $\Omega(n) \Rightarrow \Theta(n)$.

merge

- $\Theta(n)$ om man använder binära heapar implementerade med arrayer.
- Med binomialheapar: $O(\log n)$.

Binomialheapar

- Baserade på positionellt talsystem.
- $5 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2$.
- En binomialheap av storlek 5 består av två heapordnade binomialträd:
  - Ett med storlek 2^0.
  - Ett med storlek 2^2.
- Ett heapordnat binomialträd av storlek 2^k består av:
  - En rot (minst).
  - En binomialheap av storlek $2^k - 1$.

Binomialheapar

- Tom kö: Inga träd.
- `find-min`: Minsta roten.
- `merge`: ”Addera” köerna.
- `insert`: Skapa kö av storlek ett, kör `merge`.
- `delete-min`: Ta bort minsta roten, kör `merge`.
- `decrease-key`: Ändra prioritet, bubbla.

Binomialheapar

Tidskomplexiteter (värsta fallet, $O(1)$-jämförelser):

- Tom kö: $O(1)$.
- `find-min`: $O(\log n)$ ($O(1)$).
- `merge`: $O(\log n)$.
- `insert`: $O(\log n)$.
- `delete-min`: $O(\log n)$.
- `decrease-key`: $O(\log n)$
  (om positionen är känd).

Övning

Föreslå en lämplig representation av binomialheapar. Se till att `merge` är logaritmisk.

Binomialheapar

Amorterade tidskomplexiteter ($O(1)$-jämförelser):

- `merge`: $O(\log n)$.
- `insert`: $O(1)$.
- `delete-min`: $O(\log n)$.

Amorterade tidskomplexiteter

- Potential: Positiv konstant κ gånger antal träd (antal bitar $= 1$).
- Ursprungspotentialen minst (0).
- `merge`, `delete-min`: $O(\log n)$ faktisk tid, potentialen ökar med (max) $O(\log n)$.

`insert`

- `insert` tar $\Theta(z)$ steg,
  där z är positionen för den första biten $= 0$.
- Insättning ökar potentialen med $\kappa(1 - z)$.
- κ tillräckligt stor $\Rightarrow O(1)$.

Övning

Vad är tidskomplexiteten för följande program?
Antag att `i++` körs n gånger.
Använd det logaritmiska kostnadskriteriet.

```
int i = 0;
i++;
i++;
...
i++;
```

Tidskomplexiteter

	Binär heap	Binomialheap
`find-min`	$O(1)$	$O(1)$
`delete-min`	$O(\log n)$	$O(\log n)$
`insert`	$O(1)$ (medel)	$O(1)$ (amorterat)
`decrease-key`	$O(\log n)$	$O(\log n)$
`merge`	$O(n)$	$O(\log n)$

Andra prioritetsködatastrukturer

Jämförelse på Wikipedia.

Sammanfattning

- Prioritetsköer.
- Binära heapar.
- Binomialheapar.

Nästa gång:

- Hashtabeller.
- Kanske lite om grafer.