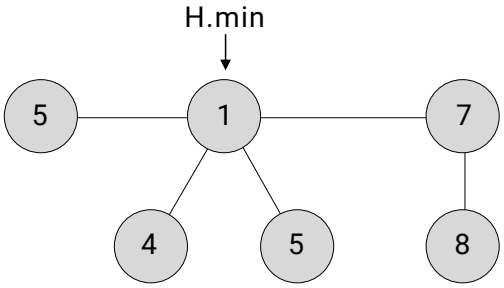
Eksamensdisposition - Fibonacci Heaps

Søren Mulvad, rbn601

3. april 2019

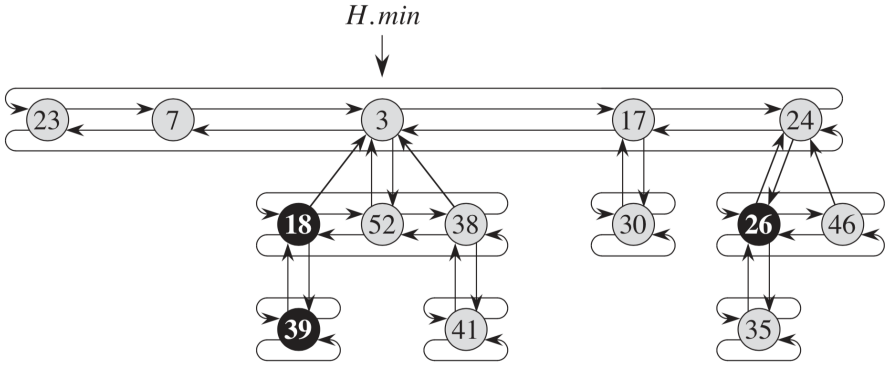
- **Motivation**
- **Struktur**
  - Egenskaber
  - Pointers/attributter
  - Operationer
- **Håndkørsel af** `Extract-Min`



- **Potentialefunktionen**
- **Amortiseret køretid for** `Extract-Min`
- **Håndkørsel af** `Decrease-Key`
- **Amortiseret køretid for** `Decrease-Key`

1 Fibonacci Heaps

- **Motivation**
  - God amortiseret køretid, mange af operationerne er i konstant tid.
  - Gør den velegnet til applikationer som benytter den mange gange. Særligt når `Extract-Min` og `Delete` bliver kaldt få gange relativt set.
  - Bl.a. Dijkstra's algoritme og Prim's algoritme (Minimum Spanning Trees) gør brug af Fibonacci Heaps.
  - Ligesom vi her vil tale om en Min-Heap kunne man ligeledes lave en Max-Heap.
- **Struktur**
  - Min-heap-egenskab: Key'en for en knude er altid $\geq$ key'en for dens forælder. En heap H har pointeren $H.min$ til minimumsknuden ved roden af træet.
  - Knuden x har pointerne/attributerne
    - $x.p$ - Forælder-knude
    - $x.child$ - En vilkårlig af børnene
    - $x.left$ og $x.right$
    - $x.mark$ - Sandhedsværdi som markerer om knuden har mistet ét barn siden den sidst blev gjort til et barn af en anden knude. Nye knuder er umarkerede, og bliver derudover umarkerede hver gang de bliver gjort til barnet af en anden knude.
    - $x.degree$ - Antallet af børn i barnelisten (f.eks. for knuden x med $x.key = 3$ i figuren nedenfor er $x.degree = 3$).
  - Når en Fibonacci Heap H er tom er $H.min = NIL$.
  - Vi bruger en cirkulær doubly linked lists til at forbinde rodknuderne og børnelisterne for en bestemt knude (se nedenfor for illustration):

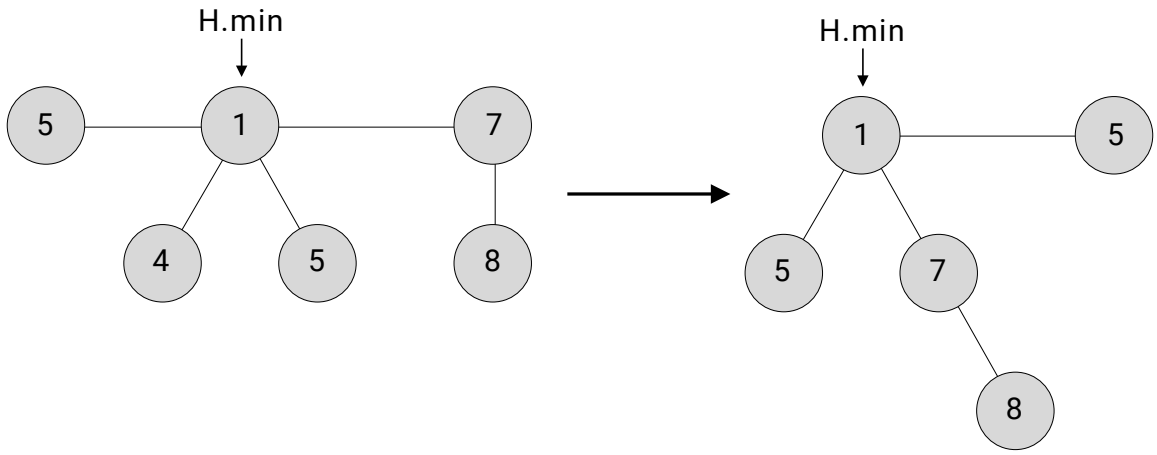


  - Operationernes amortiserede køretid og argumenter (H = heap, x = knude og k = ny key):

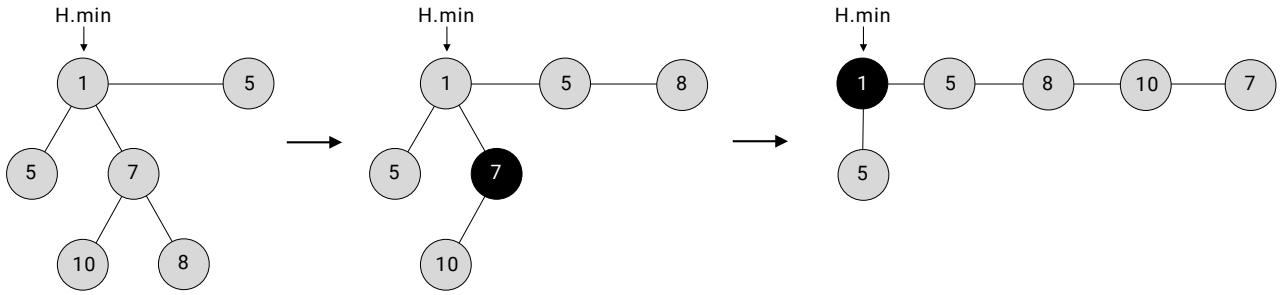
Operation	Amortiseret køretid
`Make-Heap`()	$\Theta(1)$
`Insert`(H, x)	$\Theta(1)$
`Minimum`(H)	$\Theta(1)$
`Union`(H_1, H_2)	$\Theta(1)$
`Decrease-Key`(H, x, k)	$\Theta(1)$
`Extract-Min`(H)	$\Theta(\lg n)$
`Delete`(H, x)	$\Theta(\lg n)$

  - Det tager lang tid at søge efter et element, så i ovenstående køretider antager vi at vi allerede har en pointer direkte til den knuden x vi gerne vil udføre en operation ved.

- **Håndkørsel af `Extract-Min`**
  Her er hoben før operationen og hoben efter tegnet. `Extract-Min` er det hvor vi laver et array $A[0..D(n)]$ og ser på degree af hver knude i roden.



- **Håndkørsel af `Decrease-Key`**
  Tegn en ekstra knude under 7 på det træ du kom frem til før. Fjern knuderne med key 10 og key 8. Træet før vi begynder, efter første decrease og efter andet decrease vil se således ud:



- **Potentialefunktion**
  - Definer potentialefunktionen:

$$\Phi(H) = \underbrace{t(H)}_{\text{Antal rod-knuder}} + 2\overbrace{m(H)}^{\text{Antal markerede knuder}}$$

  - Gyldighed: Vi ser at den er gyldig, da idet der er ingen knuder i vors Fib-Heap vil den være 0. Ellers kan der kun være et positivt antal knuder og/eller markerede knuder, og derfor vil den ellers altid være positiv.
  - Den amortiserede cost af enhver operation er dens faktiske cost plus ændringen i potentialet. Den totale amortiserede cost af n operationer er: *(Vi ved dette fra kapitel 17, uddyb ikke)*

$$\sum_{i=1}^{n} \hat{c}_i = \left(\sum_{i=1}^{n} c_i\right) + \Phi(D_n) - \Phi(D_0)$$

  - Antag øvre grænse $D(n) = O(\lg n)$ for den maksimale degree på enhver knude i en n-knude stor Fib-Heap. *(Kursorisk, skal ikke bevises)*

- **Bevis for amortiseret køretid for `Extract-Min` (s. 517 bund - 518)**

Faktisk cost: $O(D(n))$ fra først maksimalt at flytte $D(n)$ børn op i roden og for at initialisere vores $D(n)$ lange degree-liste A samt at finde $H.min$ til sidst.

Derudover får vi i værste fald at antallet af rod-knuder lige efter at have `Consolidate`'d er $D(n) + t(H) - 1$ hvor:

$$\underbrace{D(n)}_{\text{Børnene fra den udtrakte knude}} + \overbrace{t(H)}^{\text{De originale rodknuder}} \underbrace{-1}_{\text{Det udtrakte element}}$$

Vi ved, at vi hver eneste gang i loopet enten rykker én frem eller linker to knuder sammen, og derfor er det totale arbejde i værste fald følgende, hvor vi antager uligheden:

$$O\Big(D(n) + t(H)\Big) \leq D(n) + t(H)$$

Potentiale: Potentialet før er $t(H) + 2m(H)$
Potentialet efter er $(D(n) + 1) + 2m(H)$ (siden højest $D(n) + 1$ rødder er tilbage da det er længden af vores array A og ingen knuder bliver markeret).

Udregning:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(H') - \Phi(H) && (1)\\
&= D(n) + t(H) + \Big((D(n) + 1) + 2m(H)\Big) - \Big(t(H) + 2m(H)\Big) && (2)\\
&= 2D(n) + 1 && (3)\\
&= O(D(n))\\
&= O(\lg n)
\end{aligned}$$

Intuition: Vores cost af at udføre hver eneste link af to knuder bliver betalt for af reduktionen i potentialet der sker når et link reducerer antallet af knuder i træet med 1.

- **Bevis for amortiseret køretid for `Decrease-Key` (s. 520-522)**

Faktisk cost: Selve det at formindske vores key tager konstant tid. Herefter kan der være c kald hvor forælderknuderne går fra sand til falsk så de skal flyttes op i roden, som hver for sig tager konstant tid, hvorved vi får en køretid på c.

Potentiale: For alle c rekursive kald pånær det sidste fjerner vi markeringen for forælderknude og flytter vores nuværende knude op i roden. Altså vil der efter udførslen være følgende antal træer:

$$t(H') = t(H) + c$$

Derudover vil der højst være følgende antal markerede knuder, da $c-1$ træer blev umarkeret i processen, mens den allersidste muligvis blev markeret:

$$m(H') = m(H) - c + 2$$

Udregning: Ændringen i potentiale vil højst være:

$$\begin{aligned}
\Phi(H') - \Phi(H) &= \Big((t(H) + c) + 2(m(H) - c + 2)\Big) - \Big(t(H) + 2m(H)\Big)\\
&= \Big((t(H) + c) + 2m(H) - 2c + 4\Big) - \Big(t(H) + 2m(H)\Big)\\
&= 4 - c
\end{aligned}$$

Herved fås den amortiserede køretid til at være:

$$\hat{c}_i = c + (4 - c) = O(1)$$