



openHPI
Einführung in die Mathematik der Algorithmik
– Priority Queue Analyse

Timo Kötzing und Pascal Lenzner

Hasso-Plattner-Institut

Universität Potsdam

- wollen n Elemente dynamisch verwalten

Aufgaben:

ExtractMin

Insert

DecreaseKey

Priority Queue:

- Elemente warten in Warteschlange
- jedes Element hat Prioritätswert
- das Element mit der höchsten Priorität kommt als nächstes an die Reihe

Zwei Implementierungsvarianten:

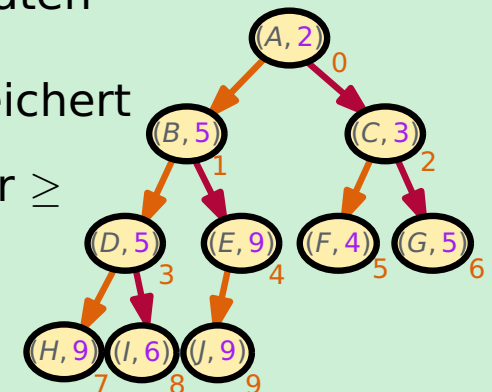
Sortierte Liste:

$(B, 2) \rightarrow (C, 5) \rightarrow (E, 8) \rightarrow (X, 27)$

- Elemente sind nach Priorität sortiert
- Direktzugriff nur auf vorderstes Element
- jedes Element kennt Vorgänger und Nachfolger (falls vorhanden)

Binärer Min-Heap:

- Elemente+Prioritäten werden in vollst. Binärbaum gespeichert
- Priorität von Vater $\geq$ Priorität Kind



Sortierte Liste mit n Elementen:

$(B, 2) \rightarrow (C, 5) \rightarrow (E, 8) \rightarrow (X, 27)$

- Elemente sind nach Priorität sortiert
- Direktzugriff nur auf vorderstes Element
- jedes Element kennt Vorgänger und Nachfolger (falls vorhanden)

Aufgaben:

ExtractMin

Insert

DecreaseKey

ExtractMin

- vorderstes Element ausgeben
- Direktzugriff auf Nachfolger als neues vorderstes Element

$\Theta(1)$

Insert

- Liste linear nach passender Stelle durchsuchen
- Element einfügen, Vorgänger und Nachfolger anpassen

$\Theta(n)$

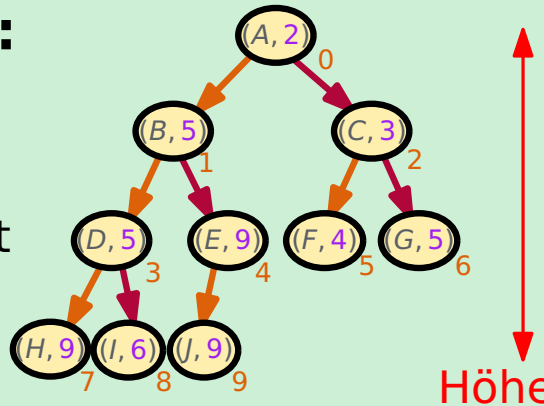
DecreaseKey

- zu änderndes Element finden
- Element entfernen, alte Nachbarn anpassen
- geändertes Element an passender Position einfügen

$\Theta(n)$

Binärer Min-Heap:

- Elemente+Prioritäten werden in vollst. Binärbaum gespeichert
- Priorität von Vater $\geq$ Priorität Kind



Aufgaben:

ExtractMin

$\Theta(\text{Höhe})$

Insert

DecreaseKey

ExtractMin

Gesamt:
 $\Theta(\text{Höhe})$

- gib Element der Wurzel zurück $\Theta(1)$
- vertausche Daten der Wurzel mit Daten des rechtesten Knotens in der letzten Ebene $\Theta(1)$
- lösche den rechtesten Knoten in der letzten Ebene $\Theta(1)$
- stelle Min-Heap-Eigenschaft ausgehend von der Wurzel wieder her

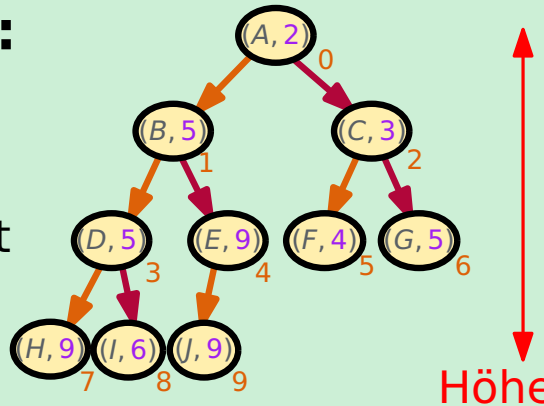
Höhe:
Anzahl Ebenen des
Min-Heaps - 1

Pro Knoten, falls Eigenschaft verletzt: Tausche mit Kind mit kleinstem Schlüssel!

$\Theta(\text{Höhe})$

Binärer Min-Heap:

- Elemente+Prioritäten werden in vollst. Binärbaum gespeichert
- Priorität von Vater $\geq$ Priorität Kind



Aufgaben:

ExtractMin

$\Theta(\text{Höhe})$

Insert

$\Theta(\text{Höhe})$

DecreaseKey

Insert

- erstelle neuen Knoten als rechtester Knoten in letzter Ebene $\Theta(1)$
- neuer Knoten erhält Element + Schlüssel $\Theta(1)$
- stelle Min-Heap-Eigenschaft ausgehend vom neuen Knoten wieder her

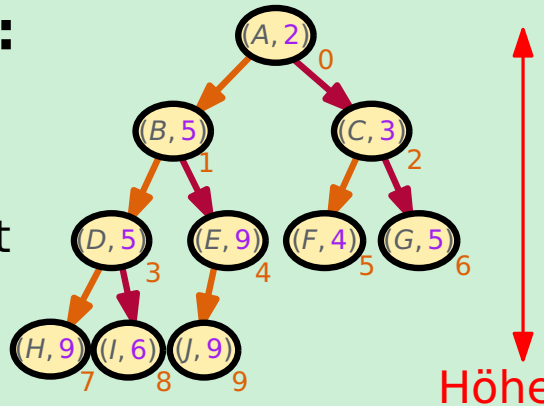
Gesamt:
 $\Theta(\text{Höhe})$

Pro Knoten, falls Eigenschaft verletzt: Tausche mit Vater!

$\Theta(\text{Höhe})$

Binärer Min-Heap:

- Elemente+Prioritäten werden in vollst. Binärbaum gespeichert
- Priorität von Vater $\geq$ Priorität Kind



Aufgaben:

ExtractMin $\Theta(\text{Höhe})$

Insert $\Theta(\text{Höhe})$

DecreaseKey $\Theta(\text{Höhe})$

DecreaseKey

■ per Direktzugriff auf zu ändernden Knoten zugreifen $\Theta(1)$

■ Schlüsselwert verringern $\Theta(1)$

Gesamt:
 $\Theta(\text{Höhe})$

■ stelle Min-Heap-Eigenschaft ausgehend vom geänderten Knoten wieder her

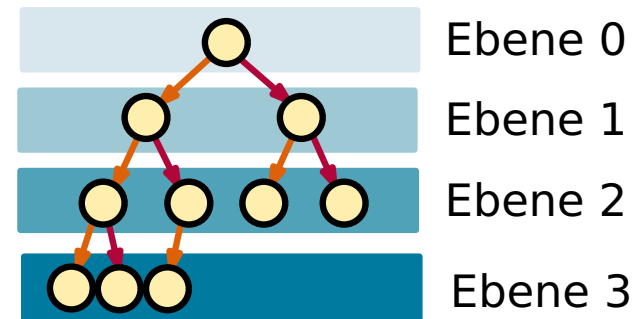
Pro Knoten, falls Eigenschaft verletzt: Tausche mit Vater!

$\Theta(\text{Höhe})$

Wie hoch kann ein binärer Min-Heap mit n Elementen maximal sein?

Wie hoch kann ein binärer Min-Heap mit n Elementen maximal sein?

- binäre Min-Heaps sind vollständige Binärbäume (außer evtl. letzte Ebene)
- in Ebene 0 genau 1 Knoten
- in Ebene 1 genau 2 Knoten
- in Ebene i genau 2^i Knoten



- Anzahl Knoten bis inklusive Ebene i :

$$\sum_{j=0}^i 2^j = 1 + 2 + 4 + 8 + \dots + 2^i = 2^{i+1} - 1$$

- sei k die vorletzte Ebene bei n Elementen:

$$2^{k+1} - 1 < n \leq 2^{k+1} - 1 + 2^{k+1} = 2^{k+2} - 1$$

$$\log_2(n + 1) - 1 \leq k + 1 < \log_2(n + 1)$$

$$\text{Höhe} = k + 1 \in \Theta(\log_2(n))$$

$$\log_2(n + 1) \leq k + 2$$

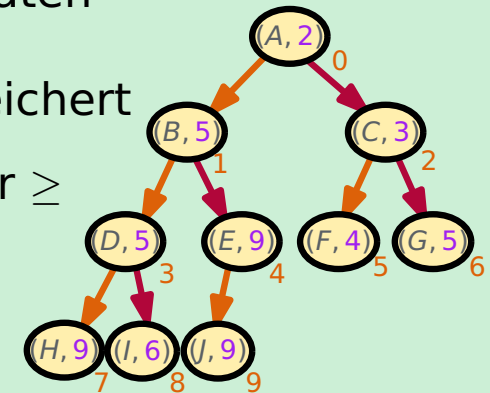
Sortierte Liste:

$(B, 2) \rightarrow (C, 5) \rightarrow (E, 8) \rightarrow (X, 27)$

- Elemente sind nach Priorität sortiert
- Direktzugriff nur auf vorderstes Element
- jedes Element kennt Vorgänger und Nachfolger (falls vorhanden)

Binärer Min-Heap:

- Elemente+Prioritäten werden in vollst. Binärbaum gespeichert
- Priorität von Vater $\geq$ Priorität Kind



- die Kosten der Operationen für n Elemente betragen im schlimmsten Fall:

ExtractMin

$\Theta(1)$

Insert

$\Theta(n)$

DecreaseKey

$\Theta(n)$

ExtractMin

$\Theta(\log_2(n))$

Insert

$\Theta(\log_2(n))$

DecreaseKey

$\Theta(\log_2(n))$