

1. Übungsblatt zu Algorithmen II im WS 2022/2023

http://algo2.iti.kit.edu/AlgorithmenII_WS22.php
{sanders, moritz.laupichler, hans-peter.lehmann}@kit.edu

Aufgabe 1 (Analyse: Kleinaufgaben)

- Geben Sie die wesentlichen Unterschiede –laut Vorlesung– zwischen einer (normalen) *Priority Queue* und einer adressierbaren *Priority Queue* an.
- Vergleichen Sie die Laufzeit einer *merge*-Operation für *Pairing Heaps* und Array basierte *Binary Heaps* (also wie aus Algorithmen I bekannt).

Aufgabe 2 (Analyse: Laufzeitverhalten)

- Beweisen Sie allgemein für adressierbare *Priority Queues* die untere Laufzeitschranke von $\Omega(\log n)$ für *deleteMin* unter der Voraussetzung, dass *insert* konstante Laufzeit benötigt.
- Warum muss diese untere Laufzeitschranke nicht gelten, wenn *insert* mehr Zeit benötigen darf?

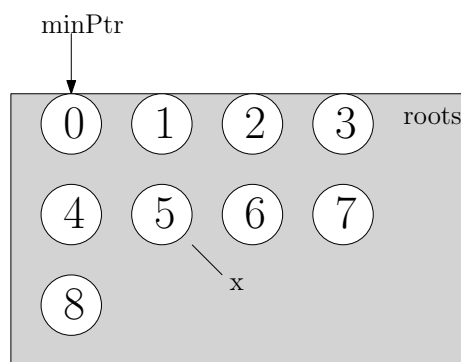
Aufgabe 3 (Analyse: best-case Verhalten)

- Geben Sie einen Zustand eines *Fibonacci Heaps* an, für den die nächsten n *deleteMin*-Operationen jeweils konstante Laufzeit benötigen (nicht amortisiert). Begründen Sie Ihre Antwort. Gehen Sie davon aus, dass zwischen den *deleteMin*-Operationen keine anderen Operationen ausgeführt werden.
- Geben Sie einen Algorithmus an, welcher den von Ihnen angegebene Zustand für beliebige n erzeugt. Beweisen Sie die Korrektheit des Algorithmus.

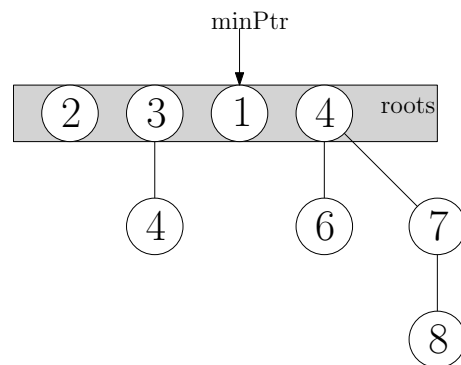
Aufgabe 4 (Rechnen: Fibonacci Heaps)

Gegeben sei ein *Fibonacci Heap* mit unten eingezeichnetem Zustand (a).

- Geben Sie eine möglichst kurze Folge von Operationen an, die diesen Zustand erzeugt.
- Führen Sie anschließend die Operationen `deleteMin()` auf dem Heap aus. Zeichnen Sie den Zustand des Heaps nach jedem Einfügen eines Baums in ein leeres Bucket und nach jeder Union-Operation.
- Geben Sie eine möglichst kurze Folge von Operationen an, die den unten eingezeichneten Zustand (b) erzeugt. Tipp: der eingezeichnete Zustand lässt sich aus dem Heap in Abbildung (a) nach der `deleteMin`-Operation durch weitere Operationen erzeugen.



(a)



(b)

Aufgabe 5 (Entwurf: Datenstrukturen)

- Erweitern Sie die Datenstruktur *Pairing Heap* um die Operation `increaseKey(h: Handle, k: Key)`. Ihre Operation sollte amortisiert $O(\log n)$ Laufzeit benötigen. Geben Sie Pseudocode an. Wie würden Sie bei einem *Binary Heap* vorgehen?
- Entwerfen Sie eine Datenstruktur welche die Operationen `insert` in $O(\log n)$, Median bestimmen in $O(1)$ und Median entfernen in $O(\log n)$ unterstützt. Eine Beschreibung in Worten ist ausreichend.

Aufgabe 6 (Rechnen: Monotone ganzzahlige Priority Queues)

Bei einer Ausführung von *Dijkstra's Algorithmus* wird folgender Ausschnitt an *Priority Queue* Operationen protokolliert:

- ...
- insert(a, 06 [00110]) (Parameter: Knotenbezeichnung, Distanz [Distanz binär])
- insert(b, 10 [01010])
- insert(c, 07 [00111])
- deleteMin()
- deleteMin()
- insert(d, 12 [01100])
- deleteMin()
- insert(e, 16 [10000])
- ...

Zusätzlich wissen Sie, dass das maximale Kantengewicht im Graphen $C = 6$ beträgt und dass vor der ersten protokollierten Operation das letzte enthaltene Element aus der *Priority Queue* entfernt wurde. Dieses hatte den Wert $min = 5$.

- a) Führen Sie die Operationen auf einer *Bucket Queue* aus. Geben Sie den Zustand der Datenstruktur nach jeder Operation an.
- b) Wieviele *Buckets* werden für eine Ausführung auf einem *Radix Heap* benötigt? Führen Sie die Operationen auf einem *Radix Heap* aus. Geben Sie den Zustand der Datenstruktur und den Wertebereich der *Buckets* nach jeder Operation an.

Aufgabe 7 (Analyse: Laufzeit von Dijkstra's Algorithmus)

Gegeben sei ein gerichteter Graph $G = (V, E)$ mit $|V| = n$ und $|E| = m$, sowie eine Kantengewichtungsfunktion $c : E \rightarrow \mathbb{R}_0^+$.

- a) Beweisen Sie die Behauptung aus der Vorlesung, dass für $m = \Omega(n \log n \log \log n)$ Dijkstra's Algorithmus mit einem *binary heap* eine durchschnittliche Laufzeit von $O(m)$ besitzt.
- b) Eine spezielle *Priority Queue* habe folgende Laufzeiteigenschaften:
- **insert:** $O(\log n)$
 - **decreaseKey:** $O(1)$
 - **deleteMin:** $O(\sqrt{m})$

(ob eine Datenstruktur mit diesen Eigenschaften existiert und Dijkstra's Algorithmus mit ihr korrekt arbeitet, ist eine andere Frage, aber wir nehmen für diese Aufgabe an es ginge :-)

Geben Sie eine kleinste obere Schranke für die Laufzeit von Dijkstra's Algorithmus unter Verwendung dieser *Priority Queue* an. Unter welcher Bedingung an das Verhältnis der Anzahl Knoten n zu Kanten m wird die Laufzeit linear in der Eingabegröße? Die Eingabe erfolgt in Form einer Adjazenzliste.

Ausgabe: 08.11.2022

Besprechung: 22.11.2022