Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Dr. Christian Schulz, Dr. Simon Gog
Michael Axtmann

1. Übungsblatt zu Algorithmen II im WS 2016/2017

http://algo2.iti.kit.edu/AlgorithmenII_WS16.php
{christian.schulz,michael.axtmann,sanders,simon.gog}@kit.edu

Aufgabe 1 (*Analyse: Kleinaufgaben*)

a) Geben Sie die wesentlichen Unterschiede –laut Vorlesung– zwischen einer (normalen) *Priority Queue* und einer adressierbaren *Priority Queue* an.

b) Vergleichen Sie die Laufzeit einer `merge`-Operation für *Pairing Heaps* und *Binary Heaps*.

Aufgabe 2 (*Analyse: Laufzeitverhalten*)

a) Beweisen Sie allgemein für adressierbare *Priority Queues* die untere Laufzeitschranke von $\Omega(\log n)$ für `deleteMin` unter der Voraussetzung, dass `insert` konstante Laufzeit benötigt.

b) Warum muss diese untere Laufzeitschranke nicht gelten, wenn `insert` mehr Zeit benötigen darf?

Aufgabe 3 (*Analyse:* best-case *Verhalten*)

a) 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.

b) Geben Sie einen Algorithmus an, welcher den von Ihnen angegebene Zustand für beliebige n erzeugt. Beweisen Sie die Korrektheit des Algorithmus.

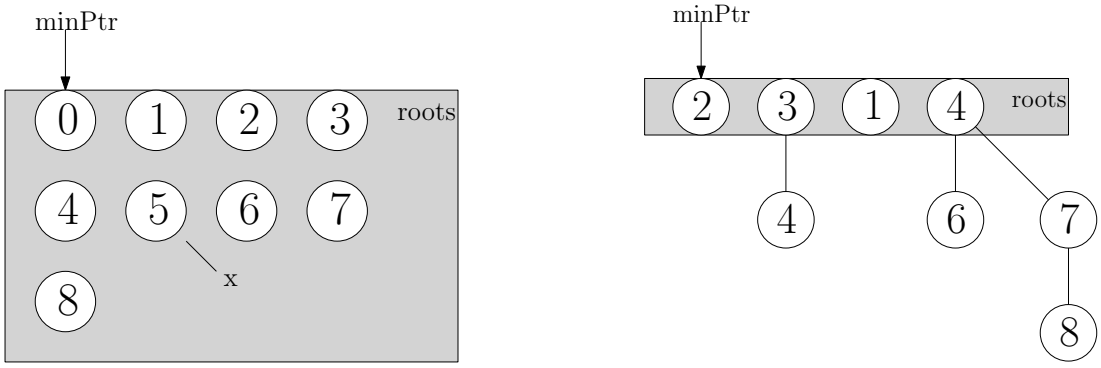
Aufgabe 4 (*Rechnen:* Fibonacci Heaps)

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

a) Geben Sie eine möglichst kurze Folge von Operationen an, die diesen Zustand erzeugt.

b) Führen Sie anschließend die Operationen `deleteMin()` auf dem Heap aus. Zeichnen Sie den Zustand des Heaps nach jedem Einfügen eines Baum in ein leeres Bucket und nach jeder Union-Operation.

c) 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*)

a) 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?

b) 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 (*Entwurf: Anwendung*)

Für ein großes –und wir meinen ein wirklich großes– Fest sind Sie für die Bar zuständig. Für diese Aufgabe haben Sie Ihren eigenen Barroboter entworfen, der automatisch wunderbare Cocktails mischen kann. Die Zutaten dafür werden in großen Kanistern bereitgestellt. Doch genau hier ist das Problem: In all der Hektik des Abends müssen Sie darauf achten, dass kein Kanister leerläuft. Sie wollen den Abend allerdings auch so gut wie möglich genießen und nicht andauernd die Kanister überprüfen. Um dieses Problem zu umgehen, gibt es nur eine Lösung: Eine Nachfüllanzeige muss her! Leider gab es nur noch Anzeigen, die es erlauben eine einzelne Zeile darzustellen.
Es ist klar, dass auf der Anzeige die am dringensten benötigte Zutat angezeigt werden sollte. Ziel ist es also einen Algorithmus zu entwerfen, der die Anzeige immer aktuell hält.

a) Überlegen Sie sich, welche Datenstruktur Sie als Grundlage für Ihren Algorithmus verwenden wollen, um ihn effizient implementieren zu können. Sie können davon ausgehen, dass die für einen Cocktail benötigten unterschiedlichen Zutaten wesentlich weniger sind als die Gesamtmenge an vorhandenen unterschiedlichen Zutaten.

b) Entwerfen Sie eine Funktion `MixDrink( recipe )`, die auf Ihrer Datenstruktur operiert. Geben Sie Pseudocode an. Sie müssen dabei nur Ihre Datenstruktur aktualisieren. Sonstige Funktionen des Roboters müssen Sie nicht berücksichtigen.

c) Wenn ein Kanister gewechselt wird, muss ihre Datenbasis natürlich auch aktualisiert werden. Beschreiben Sie, welche Auswirkungen das Wechseln auf Ihre Datenstruktur hat.

Aufgabe 7 (*Implementierung: Dijkstra's Algorithmus*)

Implementieren Sie Dijkstra's Algorithmus in `C++` mit einer adressierbaren Prioritätsliste Ihrer Wahl (https://github.com/) und der Graph-Datenstruktur aus dem KaHIP[1] Framework. Laden Sie das Archiv[2] von unserer Website mit 6 Straßennetzwerken herunter. Das Archiv beinhaltet das komplette Straßennetzwerk der USA, aber auch kleinere gewichtete Teilnetzwerke wie das Straßennetzwerk von Colorado und Florida. Die Netzwerke sind im Metis Format gespeichert. Nutzen Sie die Methoden aus dem KaHIP Framework um die Netzwerke einzulesen.

Ausgabe: 25.10.2016
Abgabe: keine Abgabe, keine Korrektur

[1] `http://algo2.iti.kit.edu/kahip`
[2] `http://algo2.iti.kit.edu/schulz/graphs.tar.gz`