Übungen zu Algorithmen | **Wintersemester 2020/21**
WSI für Informatik | **Universität Tübingen**
Kaufmann/Kuckuk/Förster | **18. November 2021**

Aufgabensammlung Teil 3

Thema: Datenstrukturen

Aufgabe 1: k-närer Bäume

In einem k-nären Baum hat jeder Knoten bis zu k Kinder. Der Wurzelknoten liegt in Level $\ell = 0$. Beantworten Sie mit kurzer Begründung:

a) Wie viele Kanten hat ein k-närer Baum der Höhe ℓ mit insgesamt n Knoten?

b) Wie viele Knoten liegen maximal in Level $\ell \geq 0$?

c) Wie viele Knoten hat ein voller Baum der Höhe ℓ insgesamt?

d) Wie viele Knoten hat ein vollständiger Baum der Höhe ℓ mindestens?

Aufgabe 2: Stacks und Queues

Angenommen Sie möchten eine Queue-Datenstruktur mit den üblichen Funktionen $\textsc{Enqueue}(\cdot)$ und $\textsc{Dequeue}(\cdot)$ bereitstellen, wozu Ihnen "intern" allerdings nur *genau 2 Stacks* zur Verfügung stehen (also keine Arrays, Listen, oder anderes).

a) Beschreiben Sie eine Implementierung der Queue (in Worten und in Pseudocode), die nur die beiden Stacks benutzt. Was können Sie über die worst-case-Laufzeit für eine $\textsc{Enqueue}(\cdot)$- bzw. $\textsc{Dequeue}(\cdot)$-Operation sagen?

b) Nun wollen wir eine Methode betrachten, mit der man die Laufzeit mehrerer Operationen gemeinsam betrachtet:

   i. Betrachten Sie eine beliebige Folge von insgesamt n_E $\textsc{Enqueue}(\cdot)$ und insgesamt n_D $\textsc{Dequeue}(\cdot)$-Operationen mit $n_E + n_D = n$.
   ii. Berechnen Sie die worst-case-Laufzeit T_n für diese Folge von Operationen in Abhängigkeit von der Anzahl der Operationen n.
   iii. Dann heißt T_n/n die *amortisierte Laufzeit* dieser Operationen.

   Falls nicht bereits geschehen, finden Sie eine Implementierung einer Queue, die wie oben nur zwei Stacks benutzt und amortisierte Laufzeit $\mathcal{O}(1)$ hat. Begründen Sie Ihre Antwort. Tipp: Betrachten Sie, welche Stack-Operationen ein Element v im schlimmsten Fall mit $\textsc{Enqueue}(\cdot)$- bzw. $\textsc{Dequeue}(\cdot)$ durchläuft.

Aufgabe 3: Heaps

Betrachten Sie die folgenden drei in Array-Schreibweise gegebenen Binärbäume (diese Reihenfolge der Indizierung heißt auch "Level-Order"):

$$T_1 = \boxed{3|6|1|7|9|2} \quad T_2 = \boxed{2|5|7|3|8|6|9} \quad T_3 = \boxed{1|4|2|7|5|9|6|8}$$

Nur einer davon (T_a) erfüllt die Min-Heap-Eigenschaft. Ein zweiter (T_b) verletzt die Min-Heap-Eigenschaft an genau einer Stelle. Der dritte (T_c) kann nur durch mehrere Operationen in einen gültigen Min-Heap umgewandelt werden.

a) Bestimmen Sie $a, b, c \in \{1, 2, 3\}$.

b) Führen Sie auf T_b die $\textsc{Heapify}$-Operation an der Fehlstelle aus. Geben Sie Zwischenschritte und das Resultat in Level-Order an.

c) Führen Sie auf T_a nacheinander die folgenden Operationen aus: Fügen Sie eine 3 hinzu mit Insert(7), extrahieren Sie das Minimum mit ExtractMin, und erhöhen Sie die 2 auf 10 mit IncreaseKey($2 \mapsto 10$). Geben Sie nach jeder Operation das Ergebnis in Level-Order an.

d) Erstellen Sie aus T_c einen Min-Heap mittels der Operation BuildMinHeap. Geben Sie das Resultat in Level-Order an.

Lösungen:

1. a) $n-1$ Kanten

   b) k^{ℓ} Knoten

   c) $\frac{k^{\ell+1}-1}{k-1}$ Knoten

   d) $\frac{k^{\ell}-1}{k-1}+1$ Knoten

2. a) Seien `in` und `out` die beiden Stacks.

Input: Zu enqueuendes Element e

```
in.push(e);
```

Algorithm 1: enqueue()

Output: Das zuerst enqueuete Element in der Queue

```
if out.is_empty() then
    while in.is_empty() ≠ ⊥ do
        out.push(in.pop());
    end
end
return out.pop();
```

Algorithm 2: dequeue()

   b) Ein Enqueue kostet immer 1. Andererseits kostet ein Dequeue $2+3k$ wobei k der Anzahl der von `in` zu `out` zu übertragenden Elemente entspricht. Nun gibt es mindestens k Enqueues zwischen zwei Dequeues der Kosten > 1. Somit kosten n Operationen im schlimmsten Fall $n-1\cdot c_e+1\cdot c_d=(n-1)\cdot 1+1\cdot(3n-1)=4n-2$. Es ergeben sich durchschnittliche Kosten von $4-2/n$.

3. a) $a=3, b=1, c=2$

   b) Heapify(3) liefert $\begin{array}{|c|c|c|c|c|c|}\hline 1&6&2&7&9&3\\\hline\end{array}$

   c) Insert(3) liefert $\begin{array}{|c|c|c|c|c|c|c|c|c|}\hline 1&3&2&4&5&9&6&8&7\\\hline\end{array}$
   darauf ExtractMin liefert $\begin{array}{|c|c|c|c|c|c|c|c|}\hline 2&3&6&4&5&9&7&8\\\hline\end{array}$ (hier wurde 7 als neue Wurzel gewählt)
   darauf DecreaseKey($2\mapsto 10$) liefert $\begin{array}{|c|c|c|c|c|c|c|c|}\hline 3&4&6&8&5&9&7&10\\\hline\end{array}$

   d) BuildMinHeap liefert $\begin{array}{|c|c|c|c|c|c|c|}\hline 2&3&6&5&8&7&9\\\hline\end{array}$