Algorithmen und Datenstrukturen
Übungszettel 1

Martin Avanzini <martin.avanzini@uibk.ac.at>
Thomas Bauereiß <thomas.bauereiss@uibk.ac.at>
Herbert Jordan <herbert@dps.uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>

16. März, zur Besprechung am 22. März

Aufgabe 1) Warteschlangen In der Vorlesung wurde als Beispiel für eine Datenstruktur der Keller vorgestellt. Eine Ähnliche und häufig verwendete Datenstruktur ist die Warteschlange (eng.: queue). Der Unterschied zwischen den beiden Strukturen besteht darin, dass beim Auslesen und Entfernen eines Elements im Fall des Kellers das zuletzt eingefügte Element entfernt wird (last in, first out — LIFO), während bei der Warteschlange das am frühesten eingefügte Element entfernt wird (first in, first out — FIFO).

Die beiden primitiven Operationen zum Einfügen und Entfernen von Elementen in bzw. aus einer Warteschlange können wie folgt deklariert werden:

Interface einer Warteschlange

```
interface Queue {
   void enqueue(int n);
   int dequeue();
}
```

1. Gegeben sei eine leere Warteschlange `q`, auf die nacheinander die folgenden Operationen angewendet werden:

   (a) `q.enqueue(1)`

   (b) `q.enqueue(2)`

   (c) `q.enqueue(3)`

   (d) `int x = q.dequeue()`

   (e) `q.enqueue(x)`

   (f) `int y = q.dequeue()`

   Geben Sie den Inhalt der Warteschlange nach jeder dieser Operationen und die Werte der Variablen `x` und `y` an.

2. Implementieren Sie eine Warteschlange mit verketteten Listen, analog zu der in der Vorlesung vorgestellten Implementierung eines Kellers mit verketteten Listen. Sie können die Implementierung in Java (siehe Vorlage auf der Proseminar-Webseite) oder Pseudocode formulieren.

3. Geben Sie die Laufzeit der beiden Methoden `enqueue` und `dequeue` Ihrer Implementierung in atomaren Operationen an.

Aufgabe 2) $\mathcal{O}$-Notation Betrachten Sie folgende Funktionen:

- $a(n) = 0.5n$
- $b(n) = n^2 + 2n$
- $c(n) = n^2 + 2$
- $d(n) = 5$
- $e(n) = n \log n$
- $f(n) = 2n^3 + 3n^2$
- $g(n) = n \log n \cdot \sqrt{n} \log n$

1. Geben Sie an, welche der Funktionen in $\mathcal{O}(n^2)$ enthalten sind.
2. Sortieren Sie die Funktionen nach ihrem asymptotischen Wachstum, d.h. bringen Sie die Funktionen in eine Reihenfolge $f_1, \ldots, f_7$, so dass $\forall i \in \{1, \ldots, 6\} : f_i(n) = \mathcal{O}(f_{i+1}(n))$.
3. Erläutern Sie den Unterschied zwischen $\mathcal{O}(n)$ und $\Omega(n)$ bzw. $\Theta(n)$ jeweils anhand einer der oben gegebenen Funktionen.

Aufgabe 3) Minimum-Maximum-Suche mit Divide & Conquer Gegeben sei ein unsortiertes Array der Größe n. Es wird ein Verfahren gesucht, was sowohl den minimalen als auch den maximalen Wert im Array bestimmt.

Der Standard-Ansatz ist es, zweimal über das Array zu iterieren, wobei im ersten Durchlauf das Minimum und im zweiten Durchlauf das Maximum bestimmt wird.

1. Wieviele Vergleichsoperationen benötigt der Standard-Ansatz? Zählen Sie hierbei ausschließlich Schlüsselvergleiche, d.h., Vergleiche zwischen Elementen des Arrays, aber nicht etwa Vergleiche von Indizes mit 0 oder n.

2. Entwerfen Sie einen Algorithmus nach dem Prinzip Divide & Conquer. Formulieren Sie den Algorithmus in Java (siehe Vorlage auf der Proseminar-Webseite) oder Pseudocode.

3. Geben Sie für den Divide & Conquer Algorithmus die Anzahl der Schlüsselvergleiche als Rekursiongleichung an, und lösen Sie diese Gleichung für Werte $n = 2^k, k \in \mathbb{N}$.

Aufgabe 4) Master Theorem Betrachten Sie folgende Rekursionsgleichungen:

- $T_1(n) = 4T_1(\frac{n}{2}) + n$
- $T_2(n) = 6T_2(\frac{n}{3}) + n^2$
- $T_3(n) = 16T_3(\frac{n}{4}) + n^2 \log(n)$
- $T_4(n) = T_4(n-1) + n$

Bestimmen Sie für jede dieser Gleichungen eine Lösung mit Hilfe des Master Theorems, oder begründen Sie, warum das Master Theorem nicht anwendbar ist.