Universität Karlsruhe
Fakultät für Informatik
ITI Wagner

1. Übungsblatt

Ausgabe: 31. Oktober 2006
Abgabe: 8. November, 15:30 Uhr im ITI Wagner (Informatik-Hauptgebäude, 3. Stock)

Die Bearbeitung in Zweiergruppen ist ausdrücklich erwünscht.
Es gibt 6 Übungsblätter. Jedes erfolgreich bearbeitete Übungsblatt gibt einen Klausurbonus von 0.5 Punkten. Die erfolgreiche Bearbeitung von allen Übungsblättern resultiert auf jeden Fall in einem Klausurbonus von einem Teilnotenschritt. Ein Übungsblatt gilt als erfolgreich bearbeitet wenn darauf mindestens die Hälfte aller Punkte erzielt wurden. Der Klausurbonus beträgt in jedem Fall maximal einen Teilnotenschritt.

Problem 1: Average-case-Laufzeit vs. Worst-case-Laufzeit 1pt

Betrachten Sie das folgende Suchproblem **lineare Suche**:

- **Eingabe:** Eine Sequenz von n ganzen Zahlen $A[1,\ldots,n]$ und ein Wert v
- **Ausgabe:** Ein Index i, mit $v = A[i]$ falls v in A vorkommt, sonst der spezielle Wert NIL

(a) Schreiben Sie ein Programm in Pseudocode, und beweisen Sie, dass Ihr Algorithmus korrekt ist.

(b) Bestimmen Sie die Average-case, und die Worst-case Laufzeit Ihres Algorithmus asymptotisch scharf. Berücksichtigen Sie dabei lediglich die Vergleichsoperationen.

(c) Geben Sie eine untere Schranke für lineare Suche an.

Problem 2: Rekursion 2pt

(a) Bestimmen Sie asymptotisch scharfe Schranken für die folgenden Rekursionsgleichungen mit Hilfe des Mastertheorems:

(i) $T(n) = 4T(n/2) + n$
(ii) $T(n) = 4T(n/2) + n^2$
(iii) $T(n) = 4T(n/2) + n^3$

(b) Betrachten Sie nun *rekursives Sortieren durch Einfügen*. Um ein Array $A[1,\ldots,n]$ zu ordnen, sortieren wir $A[1,\ldots,n-1]$ rekursiv und setzen $A[n]$ in das sortierte Feld $A[1,\ldots,n-1]$ ein. Schreiben Sie den Pseudocode für einen solchen Algorithmus. Geben Sie die Rekursionsgleichung und die asymptotisch scharfe Worst-case Laufzeit Ihres Algorithmus an.

(c) Ein Array $A[1,\ldots,n]$ enthält alle ganzen Zahlen von 0 bis n außer einer (dabei sei vereinfachend $n = 2^m - 1$ für ein gegebenes $m \in \mathbb{N}$). Die Elemente von A sind binär dargestellt, und die einzige Operation die wir nutzen können, um auf A zuzugreifen ist: *rufe das j-te Bit von $A[i]$ auf*. Schreiben Sie einen Algorithmus in Pseudocode, der die fehlende Zahl ausgibt und dessen Wort-case-Laufzeit $O(n)$ ist. Beweisen Sie ein möglichst scharfe obere Schranke für die Laufzeit Ihres Algorithmus.
Hinweis: Das Kopieren einer Zahl in ein Hilfsarray B sei in $O(1)$ möglich.

Problem 3: Amortisierte Analysen 3pt

(a) Betrachten Sie eine Datenstruktur, auf der eine Sequenz von n Operationen ausgeführt wird. Die i-te Operation hat Kosten i, wenn i eine Potenz von 2 ist, ansonsten Kosten 1. Analysieren Sie mit Hilfe der Ganzheitsmethode und der Buchungsmethode die amortisierte Laufzeit der Sequenz asymptotisch scharf.

(b) Gegeben sei ein erweiterter binärer Zähler, der neben der Funktion INCREMENT (erhöhen des Zählers um 1 auch eine Funktion RESET hat. Die Funktion RESET setzt den Zähler zurück auf Null. Der Zähler starte stets bei Null, und das Kippen eines Bits habe die Kosten 1. Zeigen Sie, wie ein Zähler so als Bitfeld implementiert werden kann, dass eine beliebige Folge von INCREMENT- und RESET- Operationen in $O(n)$ durchgeführt werden kann. Schreiben Sie für beide Operationen den Pseudocode, und führen Sie eine amortisierte Analyse mit Hilfe der Buchungsmethode.
(Tipp: Verwalten Sie einen Zeiger auf das größte 1-Bit.)

Problem 4: UNION FIND 2pt

In der Vorlesung wurde eine Prozedur UNION(i, j) angegeben, die mit *balancing* arbeitet, indem immer der Baum mit weniger Elementen an den Baum mit mehr Elementen „angehängt“ wird. Betrachten Sie nun die Prozedur UNION(i, j) mit *h-balancing*, wobei immer der Baum kleinerer Höhe an den Baum größerer Höhe angehängt wird.

(a) Geben Sie die Prozedur UNION(i, j) mit *h-balancing* in Pseudocode an. Gehen Sie analog zur Vorgehensweise bei UNION mit *balancing* davon aus, daß für die Wurzel r eines Baumes anstelle des Vorgängers die Höhe des Baumes in VOR$[r]$, mit negativem Vorzeichen versehen, gespeichert ist.

(b) In der Vorlesung wurde ein Lemma angegeben, das die Anzahl der Elemente in einem aus einer Folge von MAKESET- und UNION-Operationen mit *balancing* entstandenen Baum mit der Höhe des Baumes in Beziehung setzt. Gilt dieses Lemma noch, wenn statt UNION mit *balancing* nun UNION mit *h-balancing* angewendet wird?