Übungen zur Vorlesung Einführung in die Informatik II

Ausgabe: 2. Juli 2002 Abgabe: 8. Juli 2002, 13.00 Uhr

Die Abgabe erfolgt ausschließlich über das System eClaus! Siehe
http://www.informatik.uni-stuttgart.de/ifi/fk/lehre/ss02/info_II_02.html

Aufgabe 1: (Hashing) **schriftlich**

Es sei S_k die Zahl der Vergleiche, die benötigt werden, um einen Schlüssel in einer Hashtabelle zu finden, die Größe p hat und k Schlüssel enthält ("erfolgreiche Suche"). Vgl. Abschnitt 5.4 der Vorlesung (Folien sind im Netz).
Zeigen Sie (für $0 < \lambda < 1$):

a. (4 Punkte) $S_k \approx \frac{1}{\lambda} \cdot \ln\left(\frac{1}{1-\lambda}\right)$ mit $\lambda = \frac{k}{p+1}$

b. (1 Punkt) $S_k \leq E_{k+1} = \frac{1}{1-\lambda}$

Aufgabe 2: (Hashing) **zum Votieren**

(3 Punkte) Überlegen Sie sich, welche Probleme beim Löschen in Hashtabellen mit quadratischer Sondierung auftreten können. Entwickeln Sie Verfahren (mit und ohne Verwaltung von Zusatzinformationen), wie diese Probleme gelöst werden können. Welchen Aufwand haben Ihre Verfahren? Inwieweit lassen sich Ihre Entwürfe auf das Doppel-Hash-Verfahren übertragen?

Aufgabe 3: (Hashing) **zum Votieren**

(1 Punkt) Fügen Sie die Worte HASAN, SEBASTIAN, ACHIM, FLORIAN, BJOERN und STEFAN in eine Hashtabelle der Größe 7 (Indizes 0 bis 6). Benutzen Sie folgende einfache Hashfunktion $h(w) = (\text{Character}'\text{POS}(w_{\text{last}}) - \text{Character}'\text{POS}('A')) \bmod 7$ und quadratische Sondierung zur Kollisionsauflösung. Was stellen Sie fest? w_{last} bezeichnet den jeweils letzten Buchstaben (in dieser Aufgabe stets "N" oder "M").

Aufgabe 4: (Heapsort) **zum Votieren**

(1 Punkt) Bauen Sie einen Heap aus den Elementen 36, 80, 17, 20, 61, 46, 85 und 40 auf, indem Sie diese in der angegebenen Reihenfolge in den anfangs leeren Heap einsinken lassen. Löschen Sie anschließend nacheinander jeweils die Wurzel des Heaps, wobei jeweils das letzte Element in die Wurzel gesetzt wird und dann absinkt, bis der Heap völlig geleert ist.
Geben Sie den Heap nach jeder Einfüge- und Löschoperation an.

Aufgabe 5: (Sortierverfahren) **zum Votieren**

(2 Punkte) Sortieren Sie die Zahlenfolge 5, 8, 1, 6, 3, 4, 9, 10, 4, 5, 4, 7.

a. mit *Heapsort*.

b. mit *Quicksort*, verwenden Sie dabei als Pivotelement

   (i) das mittlere Element des betrachteten Teilfelds (Index $l + r$ div 2).

   (ii) das zweitgrößte der drei Elemente mit den Indizes l, r und $l + r$ div 2.

Geben Sie die Zahlenfolgen bei allen Zwischenschritten an.

Aufgabe 6: (Stabilität von Heapsort) **zum Votieren**

a. (1 Punkt) Worin unterscheiden sich Heapsort und Bottom-up-Heapsort?

b. (1 Punkt) Zeigen Sie anhand eines Beispiels, dass das Sortierverfahren *Heapsort* in der Implementierung wie im Skript nicht stabil ist.

c. (1 Punkt) Es gibt Beispiele, für die der Algorithmus deshalb nicht stabil ist, weil in der Prozedur `sink` (Skript Claus 6.1 Folie 9) die Vergleiche A(j)>v, v<A(j+1) und v<A(j) lauten anstatt A(j)≥v, v≤A(j+1) und v≤A(j). Erklären Sie den Unterschied (bleibt der Algorithmus korrekt?) und zeigen Sie anhand eines weiteren Beispiels, dass auch die Vergleiche in der zweiten Form nicht dazu führen, dass der Algorithmus stabil wird.

Aufgabe 7: (Quicksort) **schriftlich**

Um für das Sortierverfahren *Quicksort* eine iterative Lösung angeben zu können, ist die Verwendung eines Stacks erforderlich. Auf diesem werden in jedem Durchgang die Grenzen eines der beiden Teile des partitionierten Feldes gespeichert, während der andere Teil im nächsten Durchgang partitioniert wird.

a. (1 Punkt) Wie groß muss dieser Stack (bei geschickter Implementierung) in Abhängigkeit von der Zahl der zu sortierenden Elemente sein? Geben Sie die Größe in der O-Notation an und begründen Sie Ihre Antwort.

b. (4 Punkte) Geben Sie eine (bzgl. Stackgröße) möglichst speichereffiziente iterative Lösung für *Quicksort* in Ada 95 an. Testen Sie Ihr Programm am Beispiel aus Aufgabe 5 und geben Sie an, in welcher Reihenfolge welche Teilfelder sortiert werden.