Universität Stuttgart | Sommersemester 2003
Fakultät 5, Institut FMI | Lewandowski, Claus
Einf.i.d.Informatik II | 18. Juli 2003

Aufgabenblatt 12 | **Abgabe: 25. Juli, 20.00 Uhr**

Dieses Aufgabenblatt wird im Rahmen der Prüfungsvorbereitung „Studenten für Studenten" am 28.7. ab 9 Uhr besprochen. Die Punkte zählen zum Schein dazu. Das Niveau der Aufgaben entspricht dem der Prüfung. Pro Punkt hätten Sie in der Prüfung zur Bearbeitung je 4 Minuten Zeit (gegenüber der Prüfung sind die Punkte hier halbiert, d.h. in der Prüfung entspricht ein Punkt 2 Minuten Bearbeitungszeit).

Am Montag direkt im Anschluss an die Testklausur folgen noch 2 Blätter, die nicht mehr zum Schein zählen, aber noch einige Aufgaben auf Klausurniveau zu Info I+II bieten (ggf. erst ab Dienstag im Netz). Diese werden ebenfalls im Rahmen der Prüfungsvorbereitung am 28.7. besprochen.

1. Heapsort (mittel) (schriftlich, 1+0.5+1+1+1 Punkte)

Das Sortierverfahren Heapsort basiert auf "absteigenden Heaps", d.h., das größte Element steht in der Wurzel.

a) Heaps sind spezielle Bäume. Bei der Implementierung von Heaps benutzt man im Verfahren "Heapsort" jedoch statt einer Zeigerstruktur ein Array für die Elemente. Begründen Sie dieses Vorgehen.

b) Wie viele Elemente muss das Array mindestens umfassen, um darin einen Heap mit n Elementen abspeichern zu können?

☐ n Elemente ☐ $2n$ Elemente ☐ $n \cdot \log n$ Elemente ☐ 2^n Elemente

c) Geben Sie eine formale Beschreibung für die Heapbedingung für einen absteigenden Heap an, der in einem Array mit unterer Indexgrenze 1 gespeichert ist.

d) Die Prozedur `sink`(k,n) stellt die Heapbedingung für einen (absteigenden) Heap mit n Elementen für den Unterbaum an der Stelle k wieder her.

Gegeben ist ein Array mit den Elementen 12, 7, 23, 17, 42, 3, 5. Geben Sie an, welche Aufrufe von `sink(.,.)` Sie verwenden, um dieses Array in einen Heap zu verwandeln. Geben Sie die Aufrufe und das am Ende resultierende Array an.

e) Mit welchen Parametern wird die Prozedur `sink(.,.)` das erste Mal <u>in der Sortierphase</u> aufgerufen? Führen Sie diesen Schritt durch und geben Sie das resultierende Array an.

2. Hashing, lineares Sondieren (mittel) (schriftlich, 2 Punkte)

Wenn in einer Hashtabelle der Größe p auf Platz j eine Kollision stattfinden, wählt man beim Linearen Sondieren $G(j) = (j + c) \mod p$ in der Regel die Schrittweite $c = 1$. Hätte die Wahl von z.B. $c = 2$ einen Vorteil (was ändert sich)? Was passiert, wenn das c so gewählt wird, dass $\mathrm{ggT}(c, p) \neq 1$? (Beantworten Sie nur die Fragen, das Durchführen der Linearen Sondierung ist hier NICHT zu beschreiben)

3. Hashing, quadratisches Sondieren (leicht–mittel) (schriftlich, 2 Punkte)
In eine Hashtabelle mit 11 Einträgen (Indizes 0 bis 10) sollen die Wörter CHIARELLO (4), CLAUS (1), HOLUB (10), JEHLICKA (6), KAHL (5), LEWANDOWSKI (6), SCHILLER (5), SCHMID (2), ZIMMER (4) in der angegebenen Reihenfolge eingefügt werden (die Zahlen in Klammern geben den Hashwert an). Kollisionen sollen mit dem Verfahren der quadratischen Sondierung aufgelöst werden.

Geben Sie die ausgefüllte Tabelle, sowie für jeden Eintrag alle die im Zuge der Kollisionsauflösung berechneten Indizes an.

4. Quicksort (leicht) (schriftlich, 2.5 Punkte)
Gegeben ist ein Array mit den Elementen 12, 7, 23, 17, 42, 3, 5, 8, 22, 24. Führen Sie den Quicksort Algorithmus durch. Als Pivotelement soll dabei jeweils das links stehende Element gewählt werden. Geben Sie die rekursiven Aufrufe in der Reihenfolge, wie sie bei der Berechnung auftreten, und die Inhalte der entsprechenden Teilfelder an.

5. (Bottom-Up-)Heapsort (mittel-schwer) (schriftlich, 1+3 Punkte)
Bei Bottom-Up-Heapsort wird im Gegensatz zum normalen Heapsortverfahren zunächst der Einsinkpfad bis zum Blatt berechnet und vom Blatt aus entlang des Einsinkpfads die Stelle bestimmt, an die das Element hingehört.

a) Begründen Sie kurz, warum zu erwarten ist, dass Bottom-Up-Heapsort im Mittel weniger Vergleiche benötigt als das normale Heapsort.

b) Sind nicht alle Schlüssel verschieden (z.B. wenn Paare (Vorname,Nachname) bzgl. des Nachnamens sortiert werden), so erhält man u.U. verschiedene Reihenfolgen der Elemente. Konstruieren Sie ein Beispiel, das dieses Verhalten demonstriert.

In der Klausur hätten Sie für die obigen Aufgaben 1 Stunde Zeit gehabt (in der Klausur wird natürlich ein breiteres Spektrum an Themen abgefragt).

6. Suche nach dem k-kleinstem Element (mittel-schwer) (schriftlich, 5 Punkte)
Um das k-kleinste Element einer Folge zu bestimmen, kann man so vorgehen, dass man das Feld sortiert und dann das Element mit Index k ausgibt. Um das k-kleinste Element zu bestimmen, ist es jedoch nicht nötig die komplette Folge zu sortieren. Modifizieren Sie den Quicksortalgorithmus so, dass er zu einem Feld $A[1..n]$ und einer Zahl k, $1 \leq k \leq n$ das k-kleinste Element bestimmt. Das Feld A kann danach in beliebiger Reihenfolge vorliegen. Schreiben Sie Ihr Programm in Ada 95.

Alles weitere unter

http://www.informatik.uni-stuttgart.de/fmi/fk/lehre/ss03/info_II/default.htm