Prof. G. Zachmann
Daniel Mohr

TU Clausthal
Institut für Informatik
26. Mai 2011

Sommersemester 2011

Übungen zu Informatik II - Blatt 7

Abgabe am 02.06

Organisatorisches

- Da kommenden Donnerstag Frei ist, geben Sie die Aufgaben bitte in den Übungen, oder alternativ komplett per Email bei Christian Schnarr (**christian.schnarr@tu-clausthal.de**) bis Donnerstag, ab.

Aufgabe 1 (Heaps und Heapsort, *3 Punkte*)

1. Wie groß sind die minimale und die maximale Anzahl der Elemente in einem Heap der Höhe h? (wobei der leere Baum Höhe 0 hat)
2. Ein Max-Heap ist eine Sequenze mit der Eigenschaft: $A[parent(i)] \geq A[i]$, $parent(i) = \lfloor i/2 \rfloor$. Ist die Sequenz: 23, 17, 14, 6, 13, 14 ein Max-Heap? Falls nein, welche(s) Element(e) verletzen die Max-Heap Eigenschaft?
3. Ein Min-Heap ist eine Sequenz mit der Eigenschaft: $A[parent(i)] \leq A[i]$, $parent(i) = \lfloor i/2 \rfloor$. Ist eine Sequenz, die in aufsteigend sortierter Reihenfolge vorliegt, ein Min-Heap?

Begründen Sie jeweils Ihre Angaben!

Aufgabe 2 (Counting-Sort, *5 Punkte*)

Gehen Sie im folgenden von der Counting-Sort-Implementierung aus der Vorlesung aus.

a) Sortieren Sie die Eingabefolge A = (5,1,3,2,1,0,5) mit Counting-Sort. Geben Sie das Hilfsarray C nach Ausführung der ersten Schleife ($C[i] = |\{a_j \in A | a_j = i\}|$) und der zweiten Schleife ($C[i] = |\{a_j \in A | a_j \leq i\}|$) an.

b) Ist Counting-Sort ein stabiles Sortierverfahren? Begründen Sie Ihre Antwort.

c) Angenommen der Kopf der letzten **for**-Schleife in Counting-Sort wird geändert zu

```
for j in range( 0, len(A) ):
```

Arbeitet das modifizierte Verfahren korrekt? Ist es stabil? Begründen Sie.

Aufgabe 3 (Worst-Case-Betrachtung, *2 Punkte*)

Wie viele Schritte werden im schlechtesten Fall benötigt, um in eine anfangs leere Hashtabelle n Schlüssel einzufügen, wenn für die Kollisionsbehandlung *Chaining* verwendet wird? Wie viele Schritte benötigt man insgesamt, um nach jedem der n eingefügten Schlüssel einmal zu suchen?

Aufgabe 4 (Hashtabellen in Python, *10 Punkte*)

Schreiben Sie ein Programm, welches Hashing mit $h_{a,m}(x) = x \mod m$ implementiert. Für die Kollisionsbehandlung soll *Open Addressing* mit linearer Sondierung (`s(j,k)=j`) verwendet werden. Implementieren Sie dazu:

a) eine Klasse `TableEntry`, welche die Datenstruktur für einen Eintrag der Hashtabelle enthält,

b) eine Klasse `HashTable`, welche die Hashtabelle selbst implementiert. Die Klasse soll die Funktion `h(self,key)`, `s(self,j,k)`, `insert(self,key)` und `delete(self,key)` enthalten.

c) Schreiben Sie ein Testprogramm, welches die Hashtabelle mit gleichverteilten Zufallszahlen mit Loadfaktor α füllt und danach $2{\cdot}m$ mal abwechselnd ein neues Element in die Hashtabelle einfügt und ein Element aus der Hashtabelle löscht. Auch hier sollen je ein zufälliger Wert eingefügt und ein zufällig gewähltes Element, das in der Hashtabelle enthalten ist, gelöscht werden. Geben Sie die Anzahl der Kollisionen für `insert` und `delete` für $\alpha = 0.7$ und 0.9 aus.

d) Geben Sie außerdem nach jeder der Input-/Delete-Operationen die Anzahl der bei dieser Operation benötigten Sondierungsschritte aus. (Dies ergibt eine Art Zeitreihe; wer möchte, kann diese noch plotten, z.B. mit Hilfe von Gnuplot, Excel, oder einem anderen Tool Seiner Wahl.)