Prof. G. Zachmann
Daniel Mohr

TU Clausthal
Institut für Informatik
19. Mai 2011

Sommersemester 2011

Übungen zu Informatik II - Blatt 6

Abgabe am 26.05

Organisatorisches

- Die theoretischen Aufgaben müssen Sie donnerstags in der Vorlesung abgeben.
- Die Programmieraufgaben müssen Sie donnerstags bis spätestens 13:15 Uhr an Ihren Tutor per Email (**christian.schnarr@tu-clausthal.de**) schicken.
- Die Programmieraufgaben müssen von Ihnen in der Übung vorgeführt und erklärt werden.

Aufgabe 1 (Rekursionsbäume, *3 Punkte*)

Gegeben seien n Proben (z.B. Wasserproben). Wir nehmen an, dass man mit Hilfe eines sehr teuren Verfahrens sicher messen kann, ob das Bakterium *Baktus* in einer Probe enthalten ist. *Baktus* ist sehr selten, daher gehen wir davon aus, dass es in den meisten der n Proben *nicht* vorhanden ist. Der Einfachheit halber gehen wir auch davon aus, dass, *falls* eine Probe mit *Baktus* verseucht ist, dann auch sehr stark, d.h., es schwimmen darin sehr viele Bakterien.

Da nun das Messverfahren sehr teuer ist, suchen wir eine Methode, um Messungen einzusparen (zumindest *erwartungsgemäß*).

Sie dürfen annehmen, dass jede Probe in ausreichender Menge vorhanden ist.

Geben Sie den Algorithmus in Pseudocode oder graphischer Form mit Erläuterungen an.

Aufgabe 2 (Optimierung von Bubblesort, *3 Punkte*)

a) Charakterisieren Sie den Worst-Case bei Bubblesort-Verfahren.

b) Wie könnte man den Bubblesort-Algorithmus ändern / erweitern, so dass dieser und ähnliche Fälle schneller gelöst werden? (Wir nehmen hier an, dass die Implementierung des Bubblesort den Test auf vorzeitiges Ende enthält.)

Bemerkung: solche Optimierungen ändern nicht die asymptotische worst-case Laufzeit! Noch eine Bemerkung: in einer echten Implementierung kann es sogar vorkommen, dass die tatsächliche Laufzeit der "optimierten" Bubblesort-Version langsamer ist als die einer einfacheren Version. (Das passiert leicht dann, wenn der Compiler den erzeugten Assembler-Code stark optimiert, z.B. mit `gcc -O3`.) Oftmals kann ein Compiler leichter die ihm "bekannten" Optimierungen vornehmen wenn der Source-Code einfach strukturiert ist.

Aufgabe 3 (Quicksort, *3 Punkte*)

Nehmen Sie an, Ihr Team-Kollege hätte Quicksort implementiert, aber einen Bug im Code, der dafür sorgt, dass die Laufzeit dieser Implementierung nun

$$T(n) = 3T(\frac{n}{2}) + cn$$

ist. Welche Komplexität hätte dieser Code dann?

Aufgabe 4 (Sortieralgorithmen, *3 Punkte*)

Gegeben sei eine Liste von Namen (Vor- und Nachname): Jane Doe, Fred Doe, Fred Smith, Bill Jones, Jane Jones, Mary Smith, Phil Doe, Jane Smith.

Wir sortieren diese durch irgend einen Sortieralgorithmus bzgl. deren Nachnamen, und erhalten folgende Liste: Phil Doe, Jane Doe, Fred Doe, Bill Jones, Jane Jones, Mary Smith, Fred Smith, Jane Smith.

Nun sortieren wir diese Liste noch einmal bzgl. der Vornamen. Bei welchen Sortieralgorithmen bleibt die Reihenfolge der Nachnamen erhalten, d.h., bei welchen Sortieralgorithmen erhalten wir zum Schluss eine lexikographisch sortierte Liste?

Aufgabe 5 (Optimierung von Bubblesort, *5 Punkte*)

Beim normalen Bubblesort werden immer zwei benachbarte Elemente miteinander verglichen. Ein Element, das am falschen Platz ist, wird um eine Position näher an seinen Platz geschoben. Oftmals ist es aber viele Position von seiner Endposition entfernt.

Die Idee der Optimierung ist nun, in einem "Bubble-Durchlauf" nicht direkt benachbarte Elemente zu vergleichen (und ggf. zu vertauschen), sondern Elemente, die k Positionen weit im Array auseinander liegen. (Dadurch landen Elemente evtl. zu weit "oben", aber das wird später wieder korrigiert.) Danach wird k erniedrigt, und das Ganze beginnt von vorne.

Irgendwann ist $k = 1$, d.h., man erhält den ganz normalen Bubblesort. Diesen führt man dann solange aus, bis das Array sortiert ist.

Typischerweise startet man mit $k = 0.77n$ (abgerundet auf die nächste ganze Zahl), und erniedrigt dann k jedesmal wieder um diesen Faktor. (Achtung: Abrunden ist wichtig, sonst kann es passieren, dass man nicht bei $k = 1$ ankommt!)

Implementieren Sie den Algorithmus in Python.