Abteilung Algorithmik **Sommer '11**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Dr. Alexander Kröller

Algorithm Engineering
Übung 1 vom 18. 4. 2011

Dieses Übungsblatt umfasst Aufgaben über viele Punkte. Davon sind **maximal 10** erreichbar. Die Studienleistung ist erbracht, wenn am Semesterende durchschnittlich jeweils mindestens 5 Punkte erreicht wurden.
Es gibt theoretische und praktische Aufgaben, die verschieden bearbeitet werden:

- Theoretische Aufgaben (Tx) sollen durchgearbeitet und in der Übung **am 2. 5.** diskutiert werden. Es ist nichts abzugeben. Wer in der Übung angibt, eine Aufgabe gelöst zu haben, bekommt die volle Punktzahl für die Aufgabe, und es wird erwartet, dass er sich an der Diskussion beteiligt. Schummeln bei dieser Angabe führt zum Verlust aller Punkte auf diesem Blatt. Bei Aufgaben mit "x+y P." ist das Lösen von Teilaufgaben möglich.

- Praktische Aufgaben (Px) erfordern eine Implementierung. Favorisiert werden Abgaben in C oder C++, aber es wird auch Java und Pascal akzeptiert. Andere Sprachen nur nach Rückfrage. Implementierungen müssen **bis zum 29. 4.** per Mail an `a.kroeller@tu-bs.de` eingereicht werden.

Aufgabe T1 (Unbounded Arrays): Entwickle eine Datenstruktur, die ein unbounded Array implementiert. Sie soll init(), get(i), set(i,x), push_back(x) und pop_back() in jeweils $O(1)$ Worst-Case-Zeit ermöglichen. (Das heisst, *keine* amortisierten Kosten!)
(Hinweis: Nimm mal den Beweis #2.1 zu amortisierten Kosten wörtlich) **(2 P.)**

Aufgabe T2 (Sortieren): Wir betrachten Algorithmen, die auf Vergleichen basieren (d.h., der Algorithmus hat eine Menge $\{x_1, \ldots, x_n\}$ und eine $\leq$-Operation gegeben).

a) Zeige (d.h. beweise), dass ein Algorithmus, der auf Vergleichen basiert, mindestens $n-1$ Vergleiche durchführen muss, um das kleinste von n Elementen zu finden.

b) Zeige, dass er mindestens $n-1+\log n$ Vergleiche benötigt, um auch das Zweitkleinste zu finden.

c) Gib einen Algorithmus an, der mit $n-1+\log n$ tatsächlich auskommt.

(1+2+2 P.)

Aufgabe T3 (Sortieren): Schüler bekommen Abinoten im Bereich von 0 bis 15 Punkten.

a) Gib einen Externspeicheralgorithmus für $M = O(B)$ an, der n Schüler absteigend nach Abschlussnote sortiert, und in $O(n)$ Zeit, mit $O(n)$ I/Os, auskommt.

b) Welche Bedingungen stellst Du an M? Was passiert, wenn diese nicht erfüllt sind?

c) Wieviele I/Os sind es wirklich, ohne $O(\cdot)$-Notation? (Wenn man keine genaue Zahl angeben kann: Wieviele sind in mindestens bzw. höchstens? Und gib jeweils an, wie diese Grenzen tatsächlich erreicht werden)

(1+1+1 P.)

Aufgabe P1 (Unbounded Arrays): Falls Du Aufgabe T1 gelöst hast: Implementiere deine DS (init() und push_back(x) reichen als Operationen aus) und vergleiche deine DS mit `std::vector<>` und Stephans `vec_realloc<>` (Code wird über die VL-Homepage[1] verfügbar gemacht). **(4 P.)**

Aufgabe P2 (Ständige Aufgabe): Implementiere oder experimentiere selbständig etwas, was den Vorlesungsstoff weiter erhellt, widerlegt oder erweitert. (Offensichtlich gibt es hier ein gewisses Risiko bei der Abgabe) **(3 P.)**

[1] `http://www.ibr.cs.tu-bs.de/courses/ss11/ae/`