Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Jeremias Weihmann

Sommersemester 2012
Übungsblatt 4
14. Mai 2012

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 21. bis 25. Mai

Tutoraufgabe 1

Wir betrachten eine nichtnegative ganze Zahl $c \in \mathbb{N}_0$, repräsentiert durch ein Array der Binärdarstellung, und eine Sequenz von m Inkrement- und Dekrement-Operationen (also $c \leftarrow c + 1$ und $c \leftarrow c - 1$). Zu Anfang ist $c = 0$.

(a) Wie ist die Laufzeit einer Inkrement-Operation bzw. einer Dekrement-Operation im schlechtesten Fall in Abhängigkeit von m? Nehmen Sie (vereinfachend) an, dass jede Bitänderung Kosten von 1 verursacht.

(b) Beweisen Sie, dass die amortisierten Kosten der Inkrement-Operationen konstant sind, wenn keine Dekrement-Operationen auftreten.

(c) Geben Sie eine Sequenz von m Operationen an, so dass die Kosten in $\Theta(m \log m)$ liegen.

Tutoraufgabe 2

Betrachten Sie die Konto-Methode angewandt auf unbeschränkte Arrays für beliebige Werte α und β ($\alpha > \beta > 1$).
Zeigen Sie, dass in jedem Schritt genug Token zur Verfügung stehen, wenn bei jedem Aufruf von `pushBack` $\beta/(\beta - 1)$ Token, bei jedem Aufruf von `popBack` $\beta/(\alpha - \beta)$ Token einbezahlt werden.
Zeigen Sie dafür, dass bei jedem Aufruf von `reallocate` genügend Token vorrätig sind. Überlegen Sie sich dazu, dass `reallocate` immer mit dem Wert $\beta n'$ (für ein $n' \in \mathbb{N}$) aufgerufen wird und nach einem solchen Aufruf das Array $w = n'\beta$ Stellen hat, von denen n' belegt und $(\beta - 1)n' = ((\beta - 1)/\beta)w$ Stellen frei sind. `reallocate` wird erst wieder aufgerufen, falls $n = w$ oder $\alpha n \leq w$ ist.

Zusatzaufgabe 1

In der Vorlesung lernen wir, Algorithmen aus theoretischer Sicht zu analysieren. Neben dieser Art der Algorithmen-Analyse gibt es die experimentelle Analyse. Hierbei testet man für Eingaben das Laufzeitverhalten eines Algorithmus.
In der Abbildung 1 sind für drei Algorithmen die gemessenen oberen Schranken für die Laufzeit t in Abhängigkeit der Eingabegröße n einer solchen experimentellen Analyse angegeben. Ordnen Sie die Algorithmen nach ihrer Laufzeit.

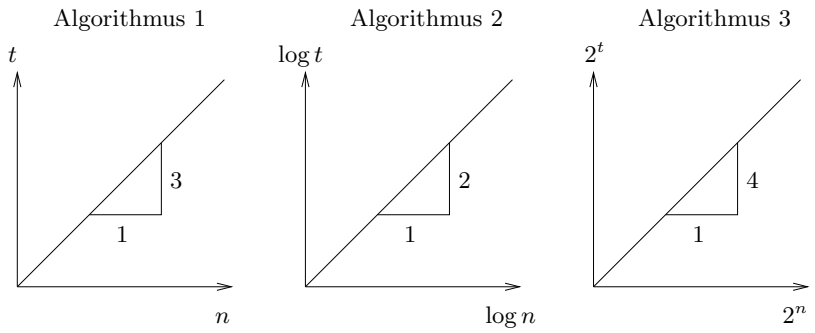

Abbildung 1: Graphen zu den Algorithmen in Aufgabe 1

Hausaufgabe 1

In der Vorlesung wurde vorgestellt, wie man ein dynamisches Array implementieren kann, sodass jede Operation amortisiert höchstens $\mathcal{O}(1)$ Zeit verbraucht. Implementieren Sie die auf der Übungswebseite bereitgestellte abstrakte Klasse `UArray` in Java in der Klasse `UIArray`. Bitte verwenden Sie in den Hausaufgaben, wie in der Vorlesung, für `alpha` bzw. `beta` die Werte 4 bzw. 2.
Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse UIArray heißt und auf den Rechnern der Rayhalle (`rayhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden.
Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an ihren Tutor.

Hausaufgabe 2

Erweitern Sie Ihre Implementierung aus Hausaufgabe 1 so zu einer Klasse `UIcArray`, dass die Operationen Ihrer Implementierung nicht nur amortisiert, sondern auch im Worst-Case, höchstens $O(1)$ Zeit benötigen.
Diese Aufgabe muss mit der Datei `mainc` kompilierbar sein.

Hinweis: Speichern Sie die Elemente in bis zu zwei Arrays. Verschieben Sie die Elemente in ein größeres Array bevor eine erneute Änderung der Größe notwendig ist.

Anmerkung: Streng genommen ist dieses Schema in Java (im Gegensatz zu einer Sprache wie c++) nicht umsetzbar, da Java bei der Erstellung eines Arrays dieses sofort mit Nullen bzw. Nullpointern füllt. Nehmen Sie also (entgegen der Realität) an, dass die Erstellung eines Arrays in Zeit $O(1)$ möglich ist, egal wie groß das Array ist.

Hausaufgabe 3

In der Vorlesung wurde angesprochen, dass ein dynamisches Array erst verkleinert werden darf, wenn das Array nur noch zur zu einem Viertel gefüllt ist. Beweisen Sie:
Wird das Array schon verkleinert, wenn der Füllstand des Arrays nur noch die Hälfte des reservierten Speicherplatzes beträgt, so stimmt die Behauptung nicht, dass jede Folge von m Operationen auf dem Array in Zeit $\mathcal{O}(m)$ abgearbeitet werden kann.