Universität des Saarlandes
FR 6.2 Informatik
Prof. Dr. Hans-Peter Lenhof
PD Dr. Peter Sanders
Dipl. Inform. Andreas Hildebrandt

5. Übung zur Vorlesung Programmierung II, SS 2003

Aufgabe 1: Unbounded Arrays, Amortisierte Analyse (10 Punkte)
Die in der Vorlesung vorgestellte Implementierung von *unbounded Arrays* verkleinert ein schon vorhandenes Array, sobald weniger als $\frac{3}{4}$ des dafür vorgesehenen Speichers verwendet wird. Für diesen Fall haben Sie gesehen, dass jede Sequenz $\sigma =< \sigma_1, \ldots, \sigma_m >$ von m `pushBack` und/oder `popBack` Operationen auf einem anfangs leeren Array mit maximaler Laufzeit von $\mathcal{O}(m)$ ausgeführt werden kann.
Gibt man statt dessen den nicht benötigten Speicher schon frei, sobald der Füllstand des Arrays auf die Hälfte des bisher vorgesehenen Platzes sinkt, so verschlechtert sich die worst-case-Laufzeit erheblich!
Zeigen Sie dies, indem Sie sich eine Sequenz von `pushBack` und `popBack` Operationen überlegen, die bei einer solchen Implementierung zu einer Laufzeit von $\Theta(m^2)$ führt.

Aufgabe 2: Binäre Zähler, Amortisierte Analyse (15 Punkte)
Eine Zahl $c \in \mathbb{N}$ kann als Array von Binärziffern d_i dargestellt werden. Auf diese Zahl soll nun eine Sequenz von m Inkrement- oder Dekrementoperationen angewendet werden, wobei zu Beginn $c = 0$ gelten soll.

(a) Bestimmen Sie die worst-case-Laufzeit einer Inkrement- oder Dekrementoperation als Funktion von m, wobei Sie in jedem Ausführungsschritt jeweils nur ein Bit verändern dürfen.

(b) Zeigen Sie: die amortisierten Kosten der Inkrementoperationen sind in $\mathcal{O}(1)$, falls in der Sequenz keine Dekrementoperationen vorkommen.

(c) Geben Sie eine Sequenz von m Inkrement- und Dekrementoperationen an, die eine Laufzeit von $\Theta(m \log(m))$ verursacht.

(d) Beschreiben Sie eine Repräsentation eines solchen Zählers, mit der Sie auch im worst-case konstante Laufzeit für Inkrement- und Dekrementoperationen garantieren können. Hierbei dürfen Sie annehmen, dass m im Voraus bekannt ist. [1]

Aufgabe 3: Implementierung einer FIFO Warteschlange (*queue*) (15 Punkte)
Implementieren Sie eine FIFO Warteschlange auf der Basis *einfach verketteter* Listen in

[1] Spannend wird es wenn Sie mit $\mathcal{O}(\log m)$ Platz auskommen. Dann ist selbst amortisiert konstante Zeit interessant.

C++. Die Operationen `pushBack`, `popFront` und `first()` sollten alle im schlimmsten Fall (worst-case) in konstanter Zeit laufen. Sie müssen keine free list verwalten, sondern können items direct mit `new` und `delete` allozieren bzw. deallozieren und Sie dürfen annehmen, dass diese Operationen in konstanter Zeit laufen.[2] Eine Beispielimplementierung einfach verketteter Listen finden Sie auf der Vorlesungshomepage.

Zusatzaufgabe 4: Unbounded Arrays (10 Bonuspunkte)
Ändern Sie die in der Vorlesung vorgestellte Datenstruktur *unbounded array* so ab, dass alle Operationen im *worst-case* in $\mathcal{O}(1)$ durchgeführt werden können. Beschreiben Sie sowohl die Datenstruktur als auch die darauf definierten Operationen und begründen Sie, warum Sie auch im schlimmsten Fall konstante Laufzeit garantieren können.

Hinweis: Verwenden Sie bis zu zwei Arrays zum Speichern der Elemente und beginnen Sie mit dem Verschieben der Elemente in ein größeres Array schon, *bevor* das kleinere Array vollständig gefüllt ist. Sie dürfen auch bei dieser Aufgabe wieder annehmen, dass Allokationen und Deallokationen in konstanter Zeit ablaufen.

Abgabe: 30.05.2003

[2] Das ist möglich aber man kann sich nicht darauf verlassen, dass reale Implementierungen dies auch wirklich leisten.