Lehrstuhl für Algorith.
und Datenstrukturen
Prof. Dr. Hannah Bast
Björn Buchhold
Claudius Korzen

Algorithmen und Datenstrukturen
(ESE + IEMS) WS 2012 / 2013

http://ad-wiki.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Übungsblatt 7

Abgabe für ESE: bis Dienstag, den 11. Dezember um 16:00 Uhr

Abgabe für IEMS: bis Dienstag, den 25. Dezember um 16:00 Uhr

Erweitern Sie die Klasse *DynamicIntArray* aus der Vorlesung. Im SVN Ordner zur Vorlesung 7 finden Sie sowohl die Java Dateien als auch eine vergleichbare Version in C++.

Aufgabe 1 (5 Punkte)

Implementieren Sie analog zu der Methode *append* aus der Vorlesung die Methode *remove*, die das letzte Element aus der Liste entfernt. Schrumpfen Sie das unterliegende fixed-size Array immer dann, wenn es nur noch zu weniger als einem Drittel gefüllt ist. Eine Folge von n append/remove Operationen sollte amortisiert lineare Laufzeit haben.

Aufgabe 2 (5 Punkte)

Schreiben Sie eine Klasse *DynamicArrayMain*, die verschiedene Tests durchführt und dabei eine Ausgabe produziert, die Sie gut in eine Kurve für die akkumulierte Laufzeit über eine Folge von n Operationen verwandeln können, z.B. wie in der Vorlesung mit *gnuplot*.

Test 1 Eine Folge aus 10 Millionen *append* Operationen auf ein anfangs leeres Feld.

Test 2 Eine Folge aus 10 Millionen *remove* Operationen auf ein Feld mit anfangs 10 Millionen Elementen.

Test 3 Eine Folge aus 10 Millionen Operationen auf ein Feld mit anfangs 1 Millionen Elementen, wobei die Operationen mit *append* anfangen und nach jeder Reallokation zwischen *append* und *remove* alternieren.

Test 4 Wie Test 3, aber beginnend mit *remove*.

Erzeugen Sie jeweils eine Kurve für die Laufzeit und committen Sie diese in einem Ordner *non-code* als Teil ihrer Lösung in das SVN.

Aufgabe 3 (5 Punkte)

Verändern Sie die Klasse so, dass nach jeder Operation gilt: *size * 1.6* $\geq$ *capacity*.

Beliebige Sequenzen aus *append* und *remove* sollten natürlich weiterhin amortisiert lineare Laufzeit haben.

Lassen Sie die Tests aus Aufgabe 2 laufen. Wenn Sie alles richtig gemacht haben, sollten die Kurven wieder einen guten Hinweis darauf geben, dass die akkumulierte Laufzeit amortisiert linear ist.

Wenn Sie die Aufgabe 3 und 4 vollständig gelöst haben, beinhaltet das praktisch auch Aufgabe 1. Sie müssen keinen Code beibehalten, der das fixed-size Array genau dann schrumpft, wenn es nur noch zu einem Drittel gefüllt ist.

Aufgabe 4 (5 Punkte) Es gibt viele Möglichkeiten, wie groß man das fixed-size Array nach dem Vergrößern oder Schrumpfen machen kann und trotzdem eine amortisiert lineare Laufzeit bekommt. Die genauen Kosten können aber variieren. Optimieren Sie Ihre Klasse dahingehend, dass weiterhin die Bedingung aus Aufgabe 3 gilt, aber diese Kosten möglichst gering werden.

Fügen Sie der Klasse einen Kostenzähler hinzu. Durch geeignete Methoden sollen die Kosten auf 0 gesetzt oder der aktuelle Stand abgefragt werden können. Jede Operation erhöht den Zähler um 1 pro *append* und *remove* und zusätzlich bei jeder Reallokation um die Anzahl der Elemente, die kopiert wurden. Im Vergleich zur Analyse aus der Vorlesung, werden hier also die Konstanten A und B in einen Topf geworfen.

Optimieren Sie Ihre Strategie so, dass die Tests aus Aufgabe 2 maximal (es geht auch etwas besser) folgende Kosten erreichen:

Test 1: 55 Mio.
Test 2: 55 Mio.
Test 3: 60 Mio.
Test 4: 55 Mio.

Dabei sollte natürlich eine Implementierung alle Tests abdecken und die Bedingung aus Aufgabe 3 weiterhin gelten. Achten Sie darauf, direkt vor den 10 Mio. Operationen, die zum Test gehören, den Zähler auf 0 zurückzusetzen.

Committen Sie, wie gehabt, Ihren Code in das SVN, in einen neuen Unterordner *uebungsblatt_07*, sowie, ebendort, Ihr Feedback in einer Textdatei *erfahrungen.txt*. Insbesondere: Wie lange haben Sie ungefähr gebraucht? An welchen Stellen gab es Probleme und wieviel Zeit hat Sie das gekostet? Wo steht das Haus vom Nikolaus?