

Übungsblatt 7

Abgabe für ESE: bis Donnerstag, den 11. Dezember um 10:00 Uhr

Abgabe für IEMS: bis Donnerstag, den 8. Januar 2015 um 10:00 Uhr

Erweitern Sie die Klasse *DynamicIntArray* aus der Vorlesung. Auf der Webseite 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)

Erweitern Sie die Klasse *DynamicArrayMain*, so dass verschiedene Tests ausgewählt werden können (am besten mit einem Kommandozeilenargument) und dabei eine Ausgabe produziert wird, die Sie gut in eine Kurve für die akkumulierte Laufzeit über eine Folge von n Operationen verwandeln können, z.B. mit *gnuplot* (s.u.)

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.

Für das Erstellen der Laufzeitkurve können Sie z.B. *gnuplot* benutzen. Lassen Sie dafür ihr Programm eine Textdatei erstellen (z.B. „*runtimes.txt*“). Am einfachsten ist es dafür die Standard-Ausgabe ihres Programms in eine Datei umzuleiten

```
./DynamicArrayMain > runtimes.txt
```

Diese Text-Datei muss folgendes Format haben:

```
<Eingabegröße 1> <TAB> <Laufzeit 1>  
<Eingabegröße 2> <TAB> <Laufzeit 2>  
...  
<Eingabegröße n> <TAB> <Laufzeit n>
```

Daraus können Sie dann mit gnuplot eine Grafik erzeugen:

```
gnuplot -e "plot 'runtimes.txt'; pause -1;"
```

Es öffnet sich ein Fenster, das den Plot enthält und den Sie dann z.B. als jpg speichern können.

Zur Erklärung: Mit „-e“ werden die angegebenen Befehle ohne den interaktiven Modus ausgeführt, und „pause -1“ verhindert, dass sich das Fenster nach dem Plotten sofort wieder schließt.

Mehr zu gnuplot gibt's mit „man gnuplot“ oder hier:

http://www.gnuplot.info/docs_4.6/gnuplot.pdf

Aufgabe 3 (7 Punkte)

Für eine Echtzeit-Anwendung benötigen Sie ein dynamisches Array mit konstanter Ausführungszeit (auch im worst-case) für alle Operationen. D.h. das lange Umkopieren bei bestimmten append() Operationen ist nicht tolerierbar. Schreiben Sie eine Klasse `WorstCaseConstantAccessTimeArray` mit den Methoden `append(int item)`, `get(int index)`, und `put(int index, int item)`, die diese Eigenschaften erfüllt, und erstellen Sie eine Laufzeit-Kurve dazu im Order *non-code*. Tipp: Speichern Sie die Elemente in bis zu zwei fixed-size Arrays und beginnen Sie, die Werte umzukopieren, bevor die Kapazität des kleineren Arrays erschöpft ist.

Für Java-Programmierer: In Java ist dieses Problem unlösbar, da es keine Möglichkeit gibt, einen Speicherbereich zu bekommen, ohne dass der vorher (mit linearer Kosten) mit Nullen gefüllt wird. Damit Sie jetzt nicht extra für diese Aufgabe auf C++ umsteigen müssen, stellen wir eine „MagicSystem“ Klasse zur Verfügung, die die Zeit während der blöden Nullenfüllaktion einfach anhält. Diese Klasse hat zwei Funktionen:

```
public class MagicSystem {  
    static long currentTimeMillis();  
    static int[] newArrayInZeroTime(int length);  
}
```

Aufgabe 4 (3 Punkte)

Finden Sie heraus, welche Vergrößerungs- und Verkleinerungsstrategie die ArrayList-Klasse (Java), bzw. der `std::vector` (C++) nutzen.

Aufgabe 5 (Zusatzaufgabe: 7 Punkte)

Der Programmierer aus dem Nachbarbüro hätte gerne ein Formel für seine Formelsammlung, mit der er ausrechnen kann, um welchen Vergrößerungsfaktor r er ein Array vergrößern muss, damit im Mittel (amortisiert) eine `append()`-Operation die Kosten $k \cdot A$ nicht übersteigt. In der Vorlesung haben wir bewiesen, dass für den Vergrößerungsfaktor $r = 3/2$ diese Kosten $4A$ sind, also $k = 4$.

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?