Lehrstuhl für Informatik VI
Prof. Dr. Frank Puppe
Am Hubland
97074 Würzburg

Dipl.-Inform. Rainer Herrler
Dipl.-Inform. Alexander Hörnlein
swt@ki.informatik.uni-wuerzburg.de

Softwaretechnik SS 2004

ÜBUNGSBLATT NR. 2
Ausgabe: Mi. 12.05.2004
Abgabe: Mi. 12.05.2004 (bis 10.00 Uhr)

VORBEMERKUNGEN:

Die JAVA Programme dieses Übungsblattes sollen auf elektronischem Wege über den sog. Praktomaten abgegeben werden. Dieser prüft schon bei der Abgabe die wichtigsten Funktionen der Programme und gibt euch im Fehlerfalle Feedback und die Möglichkeit eine verbesserte Version abzugeben. Der Praktomat wird an Montag von den Vorlesungsseiten aus erreichbar sein.
Jeder zur Vorlesung angemeldete Student hat dort ein persönliches Login (Matrikelnummer, Geburtsdatum). Je Gruppe, die ein Übungsblatt in den Briefkästen abgibt, soll nur eine Person die Programme beim Praktomaten abgeben. Diese Person ist auf dem Übungsblatt zu benennen, um die Zuordnung der Punkte zu ermöglichen.

AUFGABE 5
7 Punkte

In der Vorlesung wurde das Wechseln von Geldbeträgen als Beispiel für dynamisches Programmieren vorgestellt. Finden Sie ein weiteres Beispiel für dynamisches Programmieren (z.B. aus Literatur oder Internet) und erläutern Sie dieses knapp aber verständlich. Demonstrieren Sie dabei die Funktionsweise an einer Beispielinstanz.

AUFGABE 6 (WechselgeldRekursiv.java)
10 Punkte

Schreiben Sie die effiziente Variante zur Wechselgeldberechnung (siehe Vorlesung, Quelltext auf den Vorlesungsseiten) so um, dass sie keine Schleifenkonstrukte (for, while, ..) mehr benutzt, sondern diese durch rekursive Funktionsaufrufe ersetzt. Zeigen Sie dass auch eine Endrekursion möglich ist.
Zur Abgabe im Praktomaten muss die Lösung den Dateinamen „WechselgeldRekursiv.java“ tragen und die Methode „`public static int wechsle (int betrag)`“ enthalten.

AUFGABE 7 (Laufzeit.java)
15 Punkte (3+4+4+4)

Diese Aufgabe basiert auf Aufgabe 4 des letzten Übungsblattes. Gewünscht ist ein Programm mit dem man sehr einfach Laufzeitabschätzungen für die gegebenen Funktionen machen kann. Dazu ist in dieser Aufgabe das Programm „Laufzeit.java“ zu erstellen, in dem beliebige Funktionen als Funktionen höherer Ordnung definiert und analysiert werden können.

a) Implementieren Sie die Methode `public static int t(UnaryIntFunction f, int x)`, welche die Berechnungszeit der Funktion beim Wert x in Millisekunden angibt. Hinweise:
   a. Die Interfaceklasse UnaryIntFunction können Sie von der Vorlesungsseite herunterladen.
   b. Der Aufruf `System.currentTimeInMillis()` gibt die Systemzeit in Millisekunden an und kann für Zeitmessungen eingesetzt werden.
   c. Testen Sie die Aufgabe selbst, indem Sie in der main-Methode Funktionen von Aufgabe 4 als Instanzen der `UnaryIntFunction` definieren und bewerten lassen.

b) Erweitern Sie die Klasse um eine Methode `public static boolean laufzeitTest (UnaryIntFunction f, UnaryIntFunction o, int a, int b)`, welche entscheidet, ob für alle ganzzahligen x im Intervall [a,b]: t(f, x) <= o(x).

   Idee hinter der Aufgabestellung: Wählt man für `o` eine Standardfunktion der O-Notation (siehe Kasten) so kann man feststellen ob o(n) eine obere Grenze für die Laufzeit feststellt.

O(log n)	o(n)=c*log(n)
O(n)	o(n)=c*n

O(n log n)	o(n)=c*n *log(n)
$O(n^2)$	o(n)=c*n*n

c) Erweitern Sie die Klasse um eine Methode `public static boolean monotonFallend (UnaryIntFunction f, UnaryIntFunction o, int a, int b, int schrittweite)`, welche entscheidet, ob für alle ganzzahligen x im Intervall [a+schrittweite,b] gilt: t(f, x)/o(x) <= t(f, x-schrittweite)/o(x-schrittweite). Schreiben Sie analog eine Methode: `public static boolean monotonSteigend (UnaryIntFunction f, UnaryIntFunction o, int a, int b, int schrittweite)`.

   Idee hinter der Aufgabenstellung: Schwierigkeit bei der Verwendung von Aufgabe b ist die Bestimmung eines geeigneten Faktors c. Deshalb ermittelt die Funktion den Quotienten c = t(f,x)/o(x) und überprüft, ob dieser monoton steigend oder fallend ist. Steigen des Quotienten ist ein starker Hinweis, dass die Funktion o keine obere Schranke für die Berechnungszeit ist. Ein Fallen des Quotienten deutet dagegen darauf hin, dass die Funktion o eine obere Schranke ist.

d) Zeigen Sie unter Verwendung dieses Programms eine obere und untere Komplexitätsschranke für die Funktion aus Aufgabe 4d. Es genügt dabei zu zeigen, dass c(x) monoton fallend/steigend für einen ausgewählten Bereich ist. Geben Sie dazu in der main-Methode einen entsprechenden Aufruf an.

   Hinweis: Um die Ihre Ergebnisse unabhängiger von der Prozessorgeschwindigkeit und kleinen Geschwindigkeitsschwankungen durch das Multitasking des Betriebssystems zu machen, können Sie am Anfang eines jeden Aufrufes der Funktion f eine Millisekunde durch den Aufruf `Thread.sleep(1)` warten. Durch geeignete Wahl der Schrittweite können ebenfalls Messungenauigkeiten ausgeglichen werden.