UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

IPI: Einführung in die Praktische Informatik
Wintersemester 2018
Prof. Björn Ommer
ommer@uni-heidelberg.de

Übungsblatt 4:
Abgabe am 21.11.2018, 13:00.

Anmerkungen:

- Bei der Lösung der Programmieraufgaben dürfen ausschließlich die C++-Konstrukte verwendet werden, die bisher in der Vorlesung behandelt wurden, d.h. keine Schleifen, Variablenzuweisungen, usw. sondern ausschließlich Funktionsdefinitionen, rekursive Aufrufe und Funktionen des auf Moodle (im Verzeichnis 'C++ Quellcode') zur Verfügung gestellten Headers `fcpp.hh`, wie z.B. `cond`.
- Die Lösungen von Programmieraufgaben müssen so abgegeben werden, dass sie *ohne* das weitere Hinzufügen von Dateien oder das Einstellen von Pfaden *lauffähig* sind.

Aufgabe 1: Lambda Kalkül (4P)

Aus der Vorlesung ist Ihnen das Lambda Kalkül bekannt, das es erlaubt alle effektiv berechenbaren Funktionen darzustellen.

a) Im Folgenden sollen Sie vier Funktionen mittels des Lambda Kalküls ausdrücken. Zum Beispiel ist $\lambda x.\ 0$ eine konstante Funktion, die immer null ausgibt. Geben Sie folgende Funktionen in λ-Notation an:

1) Die Identitätsfunktion.
2) Eine Funktion, die beliebige konstante Funktionen (wie z.B. $\lambda x.\ 0$) erzeugt.
3) Eine Funktion, die als Eingabe eine Funktion f und einen Wert x erhält, und als Ausgabe die ausgewertete Funktion liefert.
4) Eine Funktion, die als Eingabe zwei Funktion g, f und eine Eingabe x erhält, und als Ausgabe zuerst f mit x und dann das Ergebnis mit g auswertet.

b) In der Vorlesung haben Sie auch die β-*Konversion* kennengelernt. Im folgenden Beispiel werden die Terme durch einfache Anwendung der β-Konversion

reduziert:

$$(\lambda x.\ x)\ (\lambda y.\ y) \Rightarrow (\lambda y.\ y)$$
$$(\lambda x.\ x\ x)\ (\lambda y.\ y) \Rightarrow (\lambda y.\ y)\ (\lambda y.\ y)$$
$$(\lambda x.\ x\ (\lambda x.\ x))\ y \Rightarrow y\ (\lambda x.\ x)$$

Ein Term ist in *β-Normalform*, wenn keine β-Konversion mehr anwendbar ist. Geben Sie für alle Terme im obigen Beispiel die *β-Normalform* an.

Nicht für jeden Term existiert eine β-Normalform. Geben Sie einen Term an für den keine β-Normalform existiert und erklären Sie, warum das der Fall ist.

Aufgabe 2: Binomialkoeffizient **(10P)**

In der Vorlesung haben Sie mit den Fibonaccizahlen ein Beispiel für eine Rekursionformel mit einem Argument n kennengelernt. Ein weiteres Beispiel ist der Binomialkoeffizient

$$\begin{aligned} B_{n,0} &= B_{n,n} = 1 && n \geq 0 \\ B_{n,k} &= B_{n-1,k-1} + B_{n-1,k} && 0 < k < n \end{aligned} \tag{1}$$

mit zwei Argumenten n und k. Dieser lässt sich grafisch in Form des Pascalschen Dreiecks veranschaulichen:

					$B_{0,0}$								1							
				$B_{1,0}$				$B_{1,1}$						1				1		
			$B_{2,0}$				$B_{2,1}$				$B_{2,2}$				1			2				1	
		$B_{3,0}$				$B_{3,1}$				$B_{3,2}$				$B_{3,3}$			1		3			3			1
$B_{4,0}$				$B_{4,1}$				$B_{4,2}$				$B_{4,3}$				$B_{4,4}$	1		4		6		4		1

Eine explizite Formel zur Berechnung des Binomialkoeffizienten ist gegeben durch

$$B_{n,k} = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2}$$

a) Schreiben Sie ein Programm, das den Binomialkoeffizient für beliebige n und k nach der in (1) gegebenen rekursive Formel berechnet. Benutzen Sie die Funktionen enter_int oder readarg_int, um n und k von der Standardeingabe einzulesen oder als Kommandozeilenparameter zu übergeben.

Benutzen Sie die Unix-Funktion time, um die Laufzeit Ihres Programms für verschiedene Kombinationen von n und k zu analysieren. Geben Sie eine

Kombination für n und k an, bei der Ihr Programm länger als 10 Sekunden für die Berechnung benötigt. Was liefert Ihr Programm für $n = 34$, $k = 18$? Können Sie dieses Ergebnis erklären?

b) In der Vorlesung wurde der (exponentielle) Rechenaufwand für die rekursive Berechnung der Fibonaccizahlen ermittelt. Versuchen Sie nun einen Ausdruck $A_{n,k}$ zur Beschreibung der Komplexität der rekursiven Berechnung des Binomialkoeffizienten zu finden.

c) Schreiben Sie nun ein Programm, das schneller arbeitet als das aus a). Benutzen Sie hierfür die explizite Formulierung aus (2). Welche asymptotische Komplexität hat Ihre schnellere Variante?

Vergleichen Sie die beiden Programme aus a) und c) hinsichtlich der Geschwindigkeit und der berechneten Ergebnisse. Was stellen Sie für höhere Werte fest? Können Sie dies erklären?

d) Überlegen Sie sich einen effizienten Algorithmus, der die ersten n Zeilen des Pascalschen Dreiecks auf den Bildschirm ausgibt. Geben Sie eine kurze Beschreibung an (keine Implementierung). Welchen Rechenaufwand hat ein solcher Algorithmus? Vergleichen Sie den Aufwand mit dem Algorithmus aus a) und versuchen Sie eine Erklärung für den Unterschied zu geben.

Aufgabe 3: Algorithmische Komplexität (6P)

a) Es seien vier Programme gegeben, deren algorithmische Komplexität $f(n)$

i) $\log n$ ii) n iii) $n \log n$ iv) n^3 v) 2^n

beträgt. Jedes Programm benötigt zur Bearbeitung von n Datensätzen 4 Sekunden auf einem Computer. Wenn sich die Menge der Datensätzen auf $2n$ verdoppelt, wie lange läuft dann jedes der vier Programme?

Wie verändert sich die Laufzeit der Programme, wenn der Computer nicht 4 Sekunden für n Datensätzen, sondern 10 oder 100 Sekunden benötigt?

b) Sortieren Sie die folgenden Funktionen danach wie schnell sie mit n wachsen. Beginnen Sie mit der langsamsten.

$n^{\log n}$ c^n $c^{(c^n)}$ $\log \log n$ n^ϵ $\log n$ 1 n^n n^c

Hier gilt $0 < \epsilon < 1 < c$.