Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Übung 6 Version: 1.0 23. 11. 2009

---

1 Mini Quiz

Die abstrakte Laufzeit eines rekursiven Algorithmus...

- ☐ lässt sich mit einer Kostenfunktion $T(n)$ abschätzen.
- ☐ kann durch mehrere Testeingaben exakt ermittelt werden.
- ☐ kann nie einer Komplexitätsklasse zugeordnet werden.
- ☐ ist der Quotient aus Eingabegröße und der Zahl der Rekursionsschritte.
- ☐ kann (ungefähr) durch die Zeilenanzahl des Codes bestimmen werden.

Komplexität:

- ☐ Die O-Notation ist generell nur für große Eingaben aussagekräftig.
- ☐ Algorithmus $A \in O(n)$ terminiert für alle Eingaben schneller als Algorithmus $B \in O(n^2)$.
- ☐ QuickSort liegt in $O(n\,log(n))$.
- ☐ $n^3 \in O(n^n)$
- ☐ $O(log(n^{1000000})) \subset O(n)$
- ☐ $0,0123456789n \in O(n)$
- ☐ $1000000000000000000000001 \in O(1)$

Akkumulatoren:

- ☐ Funktionen im Akkumulator-Stil sind immer schneller als Funktionen ohne Akkumulator.
- ☐ Manche Funktionen lassen sich nur mit Akkumulatoren implementieren.
- ☐ Ein Akkumulator wird nur benötigt für Probleme die generative Rekursion nutzen.
- ☐ Der Akkumulator-Stil führt unter Umständen zu einem Effizienzgewinn.

2 Fragen

1. Welche Schritte sind nötig, um eine Funktion im Akkumulator-Stil zu implementieren?
2. Was versteht man unter der Akkumulator-Invarianten?

3. Zeigen oder widerlegen Sie folgende Behauptung: Man kann einen Sortieralgorithmus, dessen Mechanismus auf dem Vergleich zweier Zahlen beruht, in der Komplexitätsklasse $O(log(n))$ implementieren.

   Hinweis: Dieses Problem soll Ihnen zeigen, an welche Grenzen Algorithmen manchmal stoßen. Den Beweis für die Aufgabe können Sie informell (mathematisch, zeichnerisch oder mit eigenen Worten) beschreiben. Stellen Sie sich eine Liste von Zahlen vor, die sortiert werden müssen. Wie groß ist der Aufwand, wenn jede Zahl in der Liste wenigstens einmal "angefasst" werden muss?

4. Angenommen, wir haben eine Liste mit 30 identischen Elementen. Wie verhält sich die Laufzeit von QuickSort in diesem Fall? Wäre es ratsamer, hier ausnahmsweise einen Algorithmus mit schlechterer Komplexität, beispielsweise InsertionSort, zu wählen?

5. Erklären Sie bitte in Ihren eigenen Worten, wofür $\Theta(..)$, $\Omega(..)$ und $O(..)$ stehen.

6. Welche der folgenden Aussagen gelten? Dabei sei $f(n) = 3\,n\,log_4(n)$ und $g(n) = 9\,n\,log_{10}(n)$.

   a) $O(f(n)) \subseteq \Theta(g(n))$

   b) $O(f(n)) \subseteq O(g(n))$

   c) $\Theta(f(n)) \subseteq O(g(n))$

7. Zeigen Sie, dass aus $f(n) \in O(log_a(n))$ folgt $f(n) \in O(log_b(n))$ für beliebige $a, b > 1$.

3 Rekursion vs. Iteration: Fakultät (K)

Die Fakultät von n (geschrieben $n!$) ist wie folgt definiert:

$$n! = \begin{cases} 1 & falls\, n\, =\, 0 \\ n \cdot (n-1)! & sonst \end{cases}$$

1. Implementieren Sie die Funktion `fak-rec`, die eine Zahl n konsumiert und $n!$ zurückliefert. Verwenden Sie Rekursion! Markieren Sie die Stelle in Ihrer Lösung, in der die Rekursion terminiert (Rekursionsanker) sowie die Stelle, an der die Funktion sich selbst rekursiv aufruft.

2. Veranschaulichen Sie nun Ihre Lösung (wie in der Folie T7.7), indem Sie als Parameter für die Fakultät die Zahl 6 nehmen.

3. Implementieren Sie nun die iterative Variante `fak-iter` der Fakultät unter Verwendung eines Akkumulators. Der Akkumulator soll das bisher aufmultiplizierte Produkt enthalten. Überlegen Sie sich dazu, wie der Akkumulator zu initialisieren ist!

4. Veranschaulichen Sie Ihre Lösung wieder mit der Zahl 6.

5. Vergleichen Sie beide Varianten hinsichtlich der Länge des Codes, der Komplexität der Berechnung und der Anzahl der im Speicher zu haltenden Elemente pro Rechenschritt.

4 Rekursion vs. Iteration: Fibonacci (K)

In dieser Aufgabe sollen Sie, wie in Aufgabe 3, die Fibonacci Funktion erst rekursiv, dann iterativ implementieren. Die Fibonacci-Funktion ist wie folgt definiert:

$$fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-1) + fib(n-2) & falls\ n \geq 2 \end{cases}$$

1. Implementieren Sie die Funktion `fib-rec: number -> number`, die die n-te Fibonaccizahl rekursiv berechnet. Zeichnen Sie einen Rekursionsbaum für die 4. Fibonacci Zahl und beurteilen Sie die Effizienz Ihrer Lösung. Zur Erinnerung: die ersten neun Fibonacci Zahlen (beginnend ab $n = 0$) lauten *0, 1, 1, 2, 3, 5, 8, 13 und 21*.

2. Implementieren Sie nun die Fibonacci-Funktion iterativ und beurteilen Sie einen Vorteil der iterativen Lösung gegenüber der rekursiven Lösung. Verwenden Sie Akkumulatoren in Ihrer Lösung.

5 Akkumulatoren

Lesen Sie in T8.24 über die Funktion `invert` mit Akkumulator.

1. Schreiben Sie eine Funktion `invert2`, die *ohne* Akkumulator auskommt. Vervollständigen Sie dazu folgenden Code, ohne `append` oder eine ähnliche bereits verfügbare Funktion zu verwenden:

```
;; invert2: (listof X) -> (listof X)
;; construct the reverse of list lst
;; example: (invert2 (list 'A 'B 'C)) should return (list 'C 'B 'A)
(define (invert2 lst)
    ;; rcons : (listof X) X -> (listof X)
    ;; appends el to the end of list lst.
    ;; example: (rcons (list 'A 'B) 'C) should be (list 'A 'B 'C)
   (local (
     (define (rcons lst el)
               (...) ))
     (...) ))
```

2. Vergleichen Sie Ihre Implementierung `invert2` mit `invert`. Welcher Algorithmus liegt in welcher Komplexitätsklasse?

6 Rekursion und Komplexität

Die Ackermann-Funktion ist eine 1926 von Wilhelm Ackermann gefundene mathematische Funktion, die in der theoretischen Informatik eine wichtige Rolle bezüglich Komplexitätsgrenzen von Algorithmen spielt. Sie wird als Benchmark zur Überprüfung rekursiver Prozeduraufrufe verwendet, wenn man die Leistungsfähigkeit von Programmiersprachen oder Compilern überprüfen will. Implementieren Sie die vereinfachte Ackermann-Funktion nach Péter rekursiv anhand folgender Definition:

$$\begin{aligned} Ack(0, m) &= m + 1 \\ Ack(n, 0) &= Ack(n-1, 1) \\ Ack(n, m) &= Ack(n-1, Ack(n, m-1)) \end{aligned}$$

Berechnen Sie dann den Aufruf Ack(3,4). Berechnen Sie anschließend die ersten Schritte des Aufrufs Ack(4,2). Was fällt Ihnen hierbei sehr schnell auf? Ist diese Funktion bezüglich der Komplexität von praktischem Nutzen?

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren. Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Portal auch für sie bewertet werden kann.

Abgabedatum: Freitag, 4. 12. 2009, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß der Vorgaben zu kommentieren (Scheme: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Java: JavaDoc). Zerlegen Sie Ihren Code sinnvoll und versuchen Sie, wo es möglich ist, bestehende Funktionen wiederzuverwenden. Wählen Sie sinnvolle Namen für Hilfsfunktionen und Parameter.
Benutzen sie als Sprachlevel "Zwischenstufe mit Lambda".

7 Zum Aufwärmen: Palindrome (K) (4 Punkte)

Implementieren Sie zwei Funktionen `palindrome` und `doubled-palindrome`, die eine Liste konsumieren und eine zu einem Palindrom erweiterte Liste zurückgeben. Bei der zweiten Variante soll dabei jedes Zeichen dupliziert werden. Geben Sie mindestens 2 Testfälle außer den im Template vorgegebenen Tests an. Ein Palindrom ist eine Liste, die von vorn und von hinten gelesen gleich bleibt. Dabei soll das letzte Element der Ursprungsliste *nicht* verdoppelt werden.
Beispiel: (palindrome '(a b c d)) = '(a b c d c b a), (doubled−palindrome '(1 2 3)) = (1 1 2 2 3 3 2 2 1 1)
Verwenden Sie keine explizite Rekursion, sondern `foldr` und/oder `foldl` für die Bearbeitung! Die Nutzung von *invert* bzw. *rcons* oder vergleichbarer Funktionen ist natürlich **nicht** erlaubt.

8 Optimierung der Ladeliste des ATV (6 Punkte)

Durch einen personellen Engpass haben Sie im Rahmen ihres Praktikums bei der *European Space Agency* die Aufgabe erhalten, die Ladeliste für den kommenden Flug des *Automated Transfer Vehicles (ATV)* [1] zur Internationalen Raumstation zusammenzustellen. Dabei ist es entscheidend, die verfügbare Startmasse so aufzuteilen, dass der Nutzen der Ladung (ausgedrückt durch die Dringlichkeit) maximal wird.
Die zum Transport vorgesehenen Objekte sind dabei als Liste von Scheme-Strukturen des Typs *cargo* mit ihrem Namen, der zugewiesenen Dringlichkeit (je höher, desto wichtiger) und ihrer Masse in Kilogramm gegeben.

```
1 ;; cargo represents items for transport
2 ;; name: string − the name of the cargo item
3 ;; urgency: number − the urgeny of the delivery,
4 ;; the higher, the more urgent
5 ;; mass: number − the mass of the item in kg
6 (define−struct cargo (name urgency mass))
```

Ihre Aufgabe ist es nun, aus der Liste der potentiellen Objekte diejenige Teilmenge von Objekten zu finden, die

1. in der Summe die Masse von 7.667kg (max. Nutzlastkapazität des ATV) nicht überschreitet
2. und für die es keine andere solche Menge gibt, die in der Summe ein höhere Dringlichkeit hat.

[1] siehe `http://de.wikipedia.org/wiki/Automated_Transfer_Vehicle`

Implementieren Sie zunächst eine Prozedur cargo−urgency−sum: (listof cargo) −> number, die eine Liste von Cargoelementen konsumiert und deren Gesamtdringlichkeit berechnet.
Implementieren Sie dann die Prozedur create−cargo−**list**: (listof cargo) number −> (listof cargo), die eine Liste der potenziellen Cargoelemente und die verfügbare Masse konsumiert. Als Ergebnis ist eine Liste von Cargoelementen zu produzieren, deren Gesamtgewicht unter der Maximalmasse liegt und deren Gesamtdringlichkeit ein Maximum bildet.
Hinweis: Auch wenn die maximale Last des ATV fest vorgebbar ist, müssen Sie sie als Parameter übergeben. So können auch Folgegenerationen des ATV mit Ihrem Code beladen werden.
Hinweis: Verwenden Sie Backtracking zur Lösung des Problems!