Leonhard Fellermayr
Mat.Nr. 221281305090
Tutorium Mi 14-16 (Reitmeier, HS 221 Hgb.)

WS 03/04

03.02.04

Übungen zur Vorlesung **Informatik I**

Blatt 13 - Lösungsversuch (Abgabe bis 05.02.04)

Schriftliche Aufgabe S-50

Eingabegröße n ist jeweils die Länge der zu revertierenden Liste.

In `rev_naive` (linear rekursiv) erfolgen n rekursive Aufrufe, wobei die initiale Liste jeweils vorne um ein Element gekürzt wird. Abbruchskriterium ist die leere Liste. Die Rekursion selbst hat damit Zeitaufwand $O(n)$.

Bereits traversierte Elemente der Liste werden hinter dem noch nicht traversierten Listenteil per Listenkonkatenation `@` angehängt. Am Ende muss also ein Ausdruck der Form `[]@[a]@[b]@ ... @[x]@[y]@[z]` ausgewertet werden.

Zur Auswertung eines Ausdrucks `a@b` muss allgemein zunächst die Liste `a` vollständig traversiert werden, um an ihrem Ende die Liste `b` anzuhängen. Der dazu nötige Zeitaufwand ist natürlich proportional zur Länge der Liste `a`.

Da sich am Ende eine n-fache Konkatenation von einelementigen Listen ergibt, fällt zu deren Auswertung Zeitaufwand von $\sum_{k=0}^{n} k = \frac{n(n+1)}{2} = O(n^2)$ (Gaußsche Summenformel) an.

Insgesamt hat `rev_naive` damit quadratische Zeitkomplexität.

`rev_clever` ist repetitiv rekursiv. Das jeweils erste Listenelement wird solange **vorne** an den Akkumulator `acc` angehängt, bis die Liste leer ist. Damit kommt es zu n rekursiven Aufrufen. Am Ende der Rekursion wird kein weiterer Aufwand erforderlich: die revertierte Liste steht bereits fertig im Akkumulator. Damit ist der Zeitaufwand von `rev_clever` $O(n)$.

Schriftliche Aufgabe S-51

(a) Als sinnvolle Größe der Eingabe eignet sich die Länge der Pfad-Liste l. Also: $n := l$.

(b) Die Laufzeit hängt neben der Größe von n auch noch davon ab, wo die Elemente der Liste im Baum lokalisiert sind. In der Zeile

```
h::t -> h = root b && (ispath t (left b) || ispath t (right b))
```

wird **ein** rekursiver Aufruf getätigt, falls der Teilpfad t im **linken** Teilbaum liegt, denn `ispath t (left b)` wird zu `true` ausgewertet, so dass der zweite Ausdruck `ispath t (right b)` nicht mehr berücksichtigt wird ($\rightarrow$ strikte Auswertung von OCAML). Ist dies bei allen Teilpfaden der Fall, so finden genau n Aufrufe statt.

Liegt t aber im **rechten** Teilbaum, so wird der erste Ausdruck zu `false` ausgewertet. Ein **zweiter** rekursiver Aufruf für die rechte Seite ist die Folge. Falls alle Teilpfade nach rechts verzweigen, so tritt bezüglich der Laufzeit der *worst case* ein: es müssen $2n$ Aufrufe stattfinden.

Davon abgesehen kann sich die Laufzeit aufgrund von Terminierung verkürzen,

- wenn die Höhe h des Baums kleiner ist als die Länge n der Liste
- wenn ein Element der Liste anderweitig nicht im Baum enthalten ist

Dementsprechend tritt unabhängig von n der *best case* (mit genau einem Aufruf) dann ein, wenn das erste Element der Liste l nicht der Wurzel des Baums entspricht (z. B. auch, wenn der Baum `Empty` oder die Höhe kleiner der Listenlänge ist).

(c) *best case*: Erstes Element der Liste l entspricht nicht der Wurzel des Baums. Dann ist die Zeitkomplexität von `ispath`: $O(1)$.

worst case: Alle Teilpfade der Liste l verzweigen zu rechten Teilbäumen. Dann ist die Zeitkomplexität von `ispath`: $O(2n) = O(n)$. *(Begründungen siehe oben)*

Mathematische Definition der O-Notation ([Kröger], S. 86):
Seien $\mathbb{R}^+ = x \in \mathbb{R} \mid x \geq 0$ und $f, g : \mathbb{N}_0 \to \mathbb{R}^+$.
g ist von der Ordnung f (geschrieben: $g(n)$ ist $O(f(n))$ oder auch: $g(n) = O(f(n))$, wenn es c, $n_0 \in \mathbb{N}_0$ gibt, so dass für alle $n \geq n_0$ gilt:

$$g(n) \leq c \cdot f(n).$$

Schriftliche Aufgabe S-52

Im Folgenden sei $n \in \mathbb{N}$ und $c \in \mathbb{N}_0$.

(a) Behauptung: $\log(n!) = O(n \cdot \log(n))$; dann muss gemäß Definition gelten

$$\log(n!) \leq c \cdot n \cdot \log(n)$$

$$\log(n!) \leq \log(n^n) = n \cdot \log(n) \leq c \cdot n \cdot \log(n)$$

Für $c \geq 1$ ist diese Gleichung auf jeden Fall immer erfüllt.

(b) Behauptung: $n \cdot \log(n) = O(\log(n!))$; dann muss gemäß Definition gelten

$$n \cdot \log(n) \leq c \cdot \log(n!)$$

$$n \cdot \frac{\log(n)}{\log(n!)} \leq c$$

$$c \geq n \text{ (denn } \tfrac{\log(n)}{\log(n!)} \leq 1 \text{ ist maximal 1)}$$

c lässt sich nur in Abhängigkeit von n wählen: $c \geq n$. Damit kann es kein c geben, so dass die Aussage für alle n gilt...

(c) Logarithmisches Wachstum schreitet langsamer voran als lineares oder exponentielles Wachstum ($\log(n) < n^1 < n^{1+\epsilon}$). Damit lässt sich eine Abschätzung durchführen. Für $\epsilon > 0$ ist

$$n \cdot \log n < n \cdot n^\epsilon = n^{1+\epsilon}$$

und somit auch

$$n \cdot \log n \leq c \cdot n^{1+\epsilon} \ (c \in \mathbb{N}_0, \text{ z. B. } c = 1)$$

was sich in die Definition der $O()$-Notation einsetzen lässt.

Schriftliche Aufgabe S-53

(a) Nach der obigen Definition der $O()$-Notation ist mit $c = 1$ und $f(n) := g(n)$ die Behauptung bewiesen.

Auch vom logischen Verständnis leuchtet die Behauptung sofort ein. Von der $O()$-Schreibweise wird nur gefordert, die Größenordnung der Komplexität (Komplexitäts**klasse**) anzugeben. Insofern erfolgt normalerweise eine Vergröberung des Ursprungsterms (durch Vernachlässigung von Skalaren, etc.). Der Ursprungsterm selbst ist dann natürlich weiterhin eine Teilmenge der Vergröberung.

(b) Aus den Beziehungen

$$f(n) \leq c \cdot g(n) \text{ und } g(n) \leq d \cdot h(n) \ (c, d \in \mathbb{N}_0)$$

folgt nach Transitivität

$$f(n) \leq c \cdot d \cdot O(h(n))$$

Damit ist die Behauptung bewiesen (vgl. Definition der $O()$-Notation).

(c) Es muss gelten

$$f(n) + g(n) \leq c \cdot (f(n) + g(n))$$

Wegen $x + y = 2x = 2y$ (für $x = y$) gilt $x + y < 2 \cdot \max(x, y)$ (für $x \neq y$) und infolgedessen auch

$$x + y \leq 2 \cdot \max(x, y) \text{ (für } x, y \text{ beliebig)}$$

Setzt man $x := f(n)$, $y := g(n)$, $c := 2$, so ergibt sich eine Definition der $O()$-Notation und es folgt die Behauptung.

Veranschaulichung: Die Addition zweier Polynome f (Grad n_1), g (Grad n_2) führt ebenfalls zu einem Polynom vom Grad $\max(n_1, n_2)$.