RWTH Aachen **SS 2018**
Lehrgebiet Theoretische Informatik Blatt 1
Rossmanith–Dreier–Kuinke 23.4.2018

Übung zur Vorlesung Algorithmen und Datenstrukturen

Aufgabe T1

Ordnen Sie die folgenden Funktionen in der Reihenfolge ihres asymptotischen Wachstums (ohne lange Nachzudenken). Mit $\log n$ bezeichnen wir den Logarithmus zur Basis 2.

- $\log(n)$
- $\log(n^5)/\log\log(n)$
- $n^{\log\log n}$
- $\log(n)^{\log n}$
- n^2
- $n\log n$
- 2^n

Lösungsvorschlag:
Mit $f(n) \prec g(n)$ drücken wir aus, daß $f(n) = O(g(n))$ gilt.

$$\log(n^5)/\log\log(n) \prec \log(n) \prec n\log n \prec n^2 \prec n^{\log\log n} = \log(n)^{\log n} \prec 2^n$$

Aufgabe T2

Für welche Funktionen $f\colon \mathbf{N} \to \mathbf{N}$ gilt $f(\lceil O(1)\rceil) = O(1)$? Beweisen Sie Ihre Behauptung mithilfe der Definition der O-Notation.

Lösungsvorschlag:
Es sei $f\colon \mathbf{N} \to \mathbf{N}$ eine beliebige Funktion. Wenn $g \in O(1)$, dann gibt es N und c, sodass $|g(n)| < c \cdot |1| = c$ für alle $n > N$. Das bedeutet, dass

$$\{\lceil g(n)\rceil \mid n \in \mathbf{N}\} \subseteq \{\lceil g(1)\rceil, \ldots, \lceil g(n)\rceil, -\lceil c\rceil, \ldots, \lceil c\rceil\}.$$

Wir wählen

$$C = \max\{f(\lceil g(1)\rceil), \ldots, f(\lceil g(n)\rceil), f(-\lceil c\rceil), \ldots, f(\lceil c\rceil)\}.$$

Dann ist $f(n) \leq C$ für alle natürlichen Zahlen n und damit ist $f(g(n)) = O(1)$.
Die Behauptung $f(O(1)) = O(1)$ gilt also für alle f.

Aufgabe T3

Entwerfen Sie einen einfachen Algorithmus, der zwei doppelte verkettete Listen aneinanderhängt und auf diese Weise eine neue doppelt verkettete Liste erzeugt. Können Sie dabei auf bedingte Verzweigungen verzichten?

Schreiben Sie den Algorithmus in Pseudocode oder in einer Programmiersprache nieder.

Lösungsvorschlag:

Es seien *head*1 und *head*2 die Listenköpfe der beiden Listen.

Wir müssen den Nachfolger von *head*2 an das Ende der ersten Liste hängen und dann die zyklischen Verweise reparieren. Ein Sonderfall, den wir leider beachten müssen, besteht aus einer leeren zweiten Liste. In diesem Fall müssen wir natürlich gar nichts machen.

```
if (head2.succ != head2 ) {
    head1.pred.succ = head2.succ;
    head2.succ.pred = head1.pred;
    head2.pred.succ = head1;
    head1.pred = head2.pred;
}
```

Jetzt ist *head*1 der Listenkopf der neuen Liste und *head*2 wird nicht mehr benötigt.

Aufgabe H1 (10 Punkte)

Gegeben sei ein Array $a[0], \ldots, a[n-1]$, welches Zahlen in aufsteigend sortierter Reihenfolge enthält. Entwerfen Sie einen effizienten Algorithmus, der die Anzahl der Zahlen in diesem Array bestimmt, welche sich in einem Interval $[l, r]$ befinden, wobei l und r ebenfalls zwei Zahlen sind.

Die Laufzeit ihres Algorithmus sollte $O(\log n)$ betragen.

Erläutern Sie die Arbeitsweise des Algorithmus und begründen Sie, warum er schnell ist.

Beachten Sie insbesondere den Fall, wenn das Array auch die Zahlen l und r mehrfach enthält. Auch dann muß das Ergebnis korrekt sein.

Implementieren Sie Ihren Algorithmus in einer geeigneten Sprache und lassen Sie ihn auf einigen aussagekräftigen Beispielen laufen. Verwenden Sie dabei auch Fälle mit $n = 10000000$. Überlegen Sie sich ein Verfahren, wie sie die Laufzeit ihrer Suche in diesem Falle möglichst gut messen können und geben Sie die dabei gefundene Zeit an.

Lösungsvorschlag:

Hier ist ein Programm in C:

```
#include <stdio.h>

int len(int a[], int n, int l, int r) {
  int left = 0;
  if(a[0] < l) {
    int ll=0, rr=n-1;
    while(ll <= rr) {
```

```
      int m = (ll+rr)/2;
      if(a[m] >= l) rr=m-1;
      else ll=m+1;
    }
    left = ll;
  }
  int right = n-1;
  if(a[n-1] > r) {
    int ll=0, rr=n-1;
    while(ll <= rr) {
      int m = (ll+rr)/2;
      if(a[m] <= r) ll=m+1;
      else rr=m-1;
    }
    right = rr;
  }
  return right-left+1;
}

#define N 10000000
int a[N];

int main() {
  int i;
  for(i=0; i<N; i++) {
    a[i] = i;
  }
  printf("%d\n", len(a, N, 1000, 1000));
}
```