Algorithmische Mathematik I

Wintersemester 2021
Prof. Dr. Jürgen Dölz
David Ebert

UNIVERSITÄT BONN

Blatt 5

Abgabe **Montag, 22.11.21, 10:00**.

Aufgabe 1. (Konvergenzordnung)

Sei $(x_n)_{n\in\mathbb{N}}$ eine konvergente Folge reeller Zahlen mit $x_n \to x^*$ für $n \to \infty$.

- Existiert eine Konstante $0 < c < 1$ mit
$$|x_{n+1} - x^*| \leq c|x_n - x^*| \quad , \quad \forall n > n_0,$$
so hat die Folge mindestens Konvergenzordnung 1 oder *lineare Konvergenz*.
- Gilt $|x_{n+1} - x^*| \leq c_n|x_n - x^*|$ mit $c_n \to 0$ für $n \to \infty$, so spricht man von *superlinearer Konvergenz*.
- Sei $p > 1$. Existiert eine Konstante $c > 0$, sodass
$$|x_{n+1} - x^*| \leq c|x_n - x^*|^p,$$
so hat die Folge mindestens Konvergenzordnung p.

Geben Sie für diese Folgen die Konvergenzordnung an.

a) $x_n \coloneqq (1+\epsilon)^{-n}$ mit festem $\epsilon > 0$ (geometrische Folge)

b) $x_n \coloneqq 2^{-2^{n/2}}$

c) $x_n \coloneqq 1/n^2$

d) $x_{n+1} \coloneqq \phi(x_n)$ mit $x_0 = 1/3$ und $\phi(x) = x/\ln(1/x)$

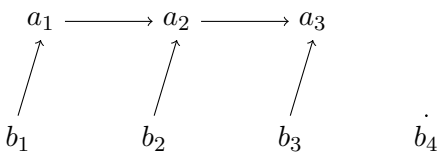
e) $x_{n+1} \coloneqq 2 + (x_n - 1)^{1/(2x_n-4)}(x_n - 2)^2$ mit $x_0 = 5/2$

(0,5+0,5+1+1+1 Punkte)

Aufgabe 2. (Optimale Suchalgorithmen)

Wir wollen Suchalgorithmen hinsichtlich der Anzahl der benötigten Vergleiche untersuchen. Sei $S(n)$ die minimale, im schlimmsten Fall benötigte Anzahl von Vergleichen unter allen möglichen Sortieralgorithmen zum sortieren einer n-elementigen Liste. In der Vorlesung wurde gezeigt, dass
$$S(n) \geq \lceil \log_2 n! \rceil.$$
Für $n = 7$ ergibt sich also $S(7) \geq 13$. Wir nehmen der Einfachheit halber an, dass alle Elemente paarweise verschieden sind.

a) Zeigen Sie per expliziter Konstruktion, dass ein Algorithmus existiert, der im schlimmsten Fall genau 13 Vergleiche benötigt. Gehen Sie dabei wie folgt vor: Zunächst werden aus den ersten sechs Elementen drei Paare gebildet und diese intern verglichen (3 Vergleiche insgesamt). Danach werden die drei größeren Elemente in die richtige Reihenfolge gebracht, indem das erste mit dem zweiten, das erste mit dem dritten, und wenn nötig das zweite mit dem dritten Element verglichen wird (im schlimmsten Fall 3 Vergleiche). Dadurch erhält man eine Situation, die durch den Graphen



beschrieben werden kann, wobei eine gerichtete Kante von Element e nach f bedeutet, dass $e < f$. Wie muss weiter vorgegangen werden, um selbst im schlimmsten Fall mit 7 zusätzlichen Vergleichen alle Elemente in die richtige Reihenfolge zu bringen?

Bemerkung: Dieser Algorithmus ist also für den schlimmsten Fall optimal hinsichtlich der benötigten Anzahl an Vergleichen, für $n = 7$. Es gibt n, für die bekannt ist, dass eine allgemeine Version dieses Algorithmus immernoch schlechter als die theoretische untere Schranke ist, man aber nicht weiß, ob ein besserer Algorithmus existiert.

b) Betrachten Sie die unsortierte Liste $(4, 1, 2, 5, 7, 6, 3)$. Nutzen Sie einmal den Algorithmus aus a) und einmal Mergesort (wie im Beispiel aus der Vorlesung), um die Liste zu sortieren. Bestimmen Sie dabei die tatsächlich benötigte Anzahl an Vergleichen. Kommentieren Sie das Ergebnis.

(3+1 Punkte)

Aufgabe 3. (Komplexitätsanalyse)

Der folgende Algorithmus 1 findet das Minimum der Einträge in einem (unsortierten) Array $A[1 \cdots n]$ $(n > 1)$.

Algorithm 1 (Minimum)

```
1: min := A[1]
2: for i ← 1 to n do
3:    if min > A[i] then
4:       min := A[i]
5:    end if
6: end for
7: return min
```

a) Wie oft wird der Test in Zeile 3 im worst case, best case und average case ausgeführt? Kann ein anderer Algorithmus zur Bestimmung des Minimums in A mit weniger Vergleichen im worst, best oder average case auskommen?

b) Wie oft wird die Zuweisung in Zeile 4 im worst, best und average case ausgeführt. *Hinweis:* Für die average case Analyse kann die folgende Beobachtung hilfreich sein: Die Zuweisung wird bei $i = k$ genau dann ausgeführt, wenn $A[k] = \min(A[1], ..., A[k])$ gilt.

(1+3 Punkte)

Aufgabe 4. (Quicksort)

Implementieren Sie Quicksort in Python um ein Array zu sortieren. Sparen Sie dabei Speicherplatz, indem Sie nur einen kleinen Zwischenspeicher zum Vertauschen benutzen. Erzeugen Sie zufällige Arrays (`numpy.random.rand`) der Länge $n = 2^0, 2^1, .., 2^{15}$ mit Einträgen aus $\{0, 1, ..., 99\}$. Implementieren Sie einen Algorithmus, der überprüft, ob das Array sortiert ist. Verfizieren Sie die asymptotische Laufzeit aus der Vorlesung, indem Sie die Laufzeit ihrer Implementierung für die Arrays messen und graphisch darstellen. Überprüfen Sie nach jeder Zeitmessung, ob Ihr Array wirklich sortiert ist und geben Sie eine Meldung aus, falls nicht.

(4 Punkte)