

Grundlagen: Algorithmen und Datenstrukturen

Woche 3

Tobias Eppacher

School of Computation, Information and Technology

12. Mai 2025

Table of contents

Aufgaben

E-Aufgaben

Hausaufgaben

Aufgabe 3.1 - Zufallsvariablen

Binäre Zufallsvariablen

- ▶ Binäre Zufallsvariable Y
- ▶ Wert von Y nur 0 oder 1
- ▶ In dieser Aufgabe: konstante Wahrscheinlichkeiten
 $\mathbb{P}(Y = 1) = c$ und $\mathbb{P}(Y = 0) = 1 - c$ $c \in [0, 1]$
- ▶ Erwartungswert: $\mathbb{E}[Y] = \sum_{y \in \{0,1\}} y * \mathbb{P}[Y = y]$
 $\mathbb{E}[Y] = 0 * \mathbb{P}[Y = 0] + 1 * \mathbb{P}[Y = 1] = c$

Erwartungswert = Summe der möglichen Werte gewichtet mit deren Auftrittswahrscheinlichkeit

Aufgabe 3.1 - Zufallsvariablen

Gegeben sei eine Zahl $a \in [0, 1]$ sowie eine Zahlenfolge (x_1, x_2, \dots, x_k) , wobei $x_i \in [0, 1]$ für alle $i \in 1, \dots, k$ gilt, wobei k die Länge der Zahlenfolge ist. Gefragt ist, ob a in der Zahlenfolge vorkommt.

Input: int a , int[] (x_1, x_2, \dots, x_k)

```
1: int  $i := 1$ 
2: while  $i \leq k$  do
3:   if  $x_i = a$  then
4:     return Ja
5:    $xi := r(xi, y)$ 
6:    $i := i + 1$ 
7: end if
8: end while
```

1. Erwartete Anzahl
Vergleiche $x_i = a$
(Alle Eingaben gleich
wahrscheinlich)

2. Asymptotisch erwartete
Laufzeit

$$\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1} \quad c \neq 1$$

Aufgabe 3.1 - Zufallsvariablen

Was ist die erwartete Anzahl an Vergleichen ($x_i = a$) bei gleicher Wahrscheinlichkeit für alle Eingaben?

Aufgabe 3.1 - Zufallsvariablen

Was ist die asymptotisch erwartete Laufzeit?

Aufgabe 3.2 - Komplexitätsanalyse

Gegeben sei eine Zahlenfolge $A[0], \dots, A[n-1]$, wobei die Länge n ist, und eine Zahl x aus dieser Folge. Gehen Sie im Folgenden davon aus, dass die Wahrscheinlichkeit von Duplikaten in A vernachlässigbar gering ist, das heißt ihre Anzahl ist im Durchschnitt in $\mathcal{O}(1)$.

```
// count(x, A)
int c = 0;
for (int i = 0; i < n; i++)
    if (A[i] == x) c++;
return c;
```

Aufgabe 3.2 - Komplexitätsanalyse (a)

Bestimmen Sie die Komplexität von `count` im worst-case, best-case und average-case.

```
// count(x, A)
int c = 0;
for (int i = 0; i < n; i++)
    if (A[i] == x) c++;
return c;
```


Aufgabe 3.2 - Komplexitätsanalyse (b)

Überlegen Sie sich einen alternativen Algorithmus, der auf **sortierten** Folgen arbeitet, und bestimmen Sie dessen Komplexität im worst-case, best-case und average-case. Vergleichen Sie diese mit den Ergebnissen aus Aufgabenteil a).

Aufgabe 3.2 - Komplexitätsanalyse (b)

```
// count(x, A) mit early stopping
int c = 0;
for (int i = 0; i < n; i++) {
    if (A[i] == x) { c++ };
    else { if (A[i] > x) return c; }
}
return c;
```

Aufgabe 3.2 - Komplexitätsanalyse (b)

```
// count(x, A) mit binärer Suche
int c = 0, l = 0, r = n - 1, m;
while (l <= r) {
    m = (l + r) / 2;
    if (A[m] == x) {
        while (m != -1 && A[m] == x) {
            m--; c++;
        }
        m = (l + r) / 2 + 1;
        while (m != A.length && A[m] == x) {
            m++; c++;
        }
        return c;
    }
    if (A[m] < x) l = m + 1;
    else r = m - 1;
}
return c;
```

Aufgabe 3.2 - Komplexitätsanalyse (b)

Andere Vorschläge? (wenn genug Zeit)

Aufgabe 3.2 - Komplexitätsanalyse (c)

Lässt sich die Komplexität verbessern, indem man die Folge zunächst sortiert?

Aufgabe 3.3 - Selbstorganisierende Liste

```
public class SelfOrganizingList<T> {  
    private static class Node<T> {  
        T data;  
        Node<T> next;  
  
        Node(T d) {  
            data = d;  
            next = null;  
        }  
    }  
  
    public void add(T data) {  
        // TODO: Neuer Knoten am Listenende + O(1) Laufzeit  
    }  
  
    public Optional<T> findFirst(Predicate<T> p) {  
        // TODO: return das erste p.test(n.data) == true  
    }  
  
    public void removeDuplicates() {  
        // TODO: Behalte erste Vorkommen von Elementen + O(1) Speicher  
    }  
}
```

Aufgabe 3.3 - Selbstorganisierende Liste (a)

```
public void add(T data) {
```

```
}
```

Aufgabe 3.3 - Selbstorganisierende Liste (b)

```
public Optional<T> findFirst(Predicate<T> p) {
```

```
}
```


Aufgabe 3.3 - Selbstorganisierende Liste (c)

```
public void removeDuplicates () {
```

```
}
```

Aufgabe 3.4 - Algorithmen und Laufzeiten

Entwerfen Sie im Folgenden zwei einfache Funktionen und erarbeiten Sie korrekten `java`-Code.

Bestimmen Sie dann die asymptotische Laufzeit Ihres Algorithmus.

Sie dürfen in ihren Algorithmen ausschließlich Schleifen, If-Statements, Additionen, Subtraktion sowie Multiplikation verwenden.

- a. Ganzzahldivision von zwei `int` $a > 0$ und $b > 0$
- b. Rest für Division (Modulo) von zwei `int` $a > 0$ und $b \neq 0$

Aufgabe 3.4 - Algorithmen und Laufzeiten (a)

Ganzzahldivision von zwei **int** $a > 0$ und $b > 0$

Aufgabe 3.4 - Algorithmen und Laufzeiten (b)

Rest für Division (Modulo) von zwei **int** $a > 0$ und $b \neq 0$

E-Aufgaben

- ▶ Aufgabe 3.5 - Noch mehr Spaß mit \mathcal{O}
 - ▶ Gute Überprüfung für Verständnis von \mathcal{O}
 - ▶ Kleiner Induktionsbeweis
- ▶ Aufgabe 3.6 - Zufallsvariablen-Caching
 - ▶ Übung zu Zufallsvariablen

Hausaufgaben

- ▶ Hausaufgabe 3 - Dynamisches Array
(Deadline: 21.05.2025)

Fragen?

- ▶ Nach Übung gerne bei mir melden
- ▶ Tutoriumschannel oder DM an mich auf Zulip
- ▶ Vorlesungschannels von GAD auf Zulip (insbesondere bei Hausaufgaben)

Feedback oder Verbesserungsvorschläge?

Gerne nach dem Tutorium mit mir quatschen oder DM auf Zulip

Bis nächste Woche!