Tutorium
Grundlagen: Algorithmen und Datenstrukturen

Übungsblatt Woche 3

Aufgabe 3.1

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

Algorithmus 1 löst diese Aufgabe. Berechnen Sie die erwartete Anzahl der Vergleiche $x_i = a$, die dieser Algorithmus durchführt, wenn jede Eingabe mit den oben beschriebenen Eigenschaften mit derselben Wahrscheinlichkeit auftritt. Bestimmen Sie zudem die asymptotisch erwartete Laufzeit.

```
Input: int a, int[] (x_1, x_2, ..., x_k)
int i = 1
while i ≤ k do
    if x_i = a then
        return Ja
    end
    i = i + 1
end
return Nein
```

Aufgabe 3.1

Zu bestimmen:

- Erwartete Anzahl an Vergleichen: $x_i = a$
- Erwartete asymptotische Laufzeit

Hinweis: Verwenden Sie, wie in der Vorlesung demonstriert, binäre Zufallsvariablen (sogenannte Indikatorvariablen). Verwenden Sie zudem, dass für eine binäre Zufallsvariable Y mit $\mathbb{P}[Y=1] = c$ und $\mathbb{P}[Y=0] = 1-c$ für ein $c \in [0,1]$, der Erwartungswert von Y genau c ist, d.h. $\mathbb{E}[Y] = \sum_{y \in 0,1} y \cdot \mathbb{P}[Y=y] = 1 \cdot \mathbb{P}[Y=1] + 0 \cdot \mathbb{P}[Y=0] = \mathbb{P}[Y=1] = c$. Sie können weiter von der Gleichung $\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}$ für $c \neq 1$ Gebrauch machen.

X_i

$$\Pr\{X_i = 1\} = \frac{2^{k-(i-1)}}{2^k} = \frac{1}{2^{i-1}}$$

$$\underline{\underline{X}} = \sum_{i=1}^{k} X_i \qquad X \text{ Anzahl Vergleiche}$$

$$\mathbb{E}\{X\} = \mathbb{E}\left[\sum_{i=1}^{k} x_i\right] = \sum_{i=1}^{k} \mathbb{E}\{x_i\} = \sum_{i=1}^{k} \frac{1}{2^{i-1}} = \sum_{i=0}^{k-1} \frac{1}{2^i} = \frac{\left(\frac{1}{2}\right)^{n+1} - 1}{\frac{1}{2} - 1}$$

$$= \frac{1 - \left(\frac{1}{2}\right)^k}{1/2} = \underbrace{2 - \left(\frac{1}{2}\right)^{k-1}}_{\text{beschränkt durch 2}}$$

$k+1 = \underline{n}$

c, d konstant

$\mathbb{E}\{\text{Laufzeit}\}$

$\mathbb{E}\{\text{Eingabe} + \text{konstanter Aufwand} + d \cdot \#\text{Vergleiche}\}$

$\mathbb{E}\{\text{Eingabe}\} + \mathbb{E}\{\text{konstant}\} + \mathbb{E}\{d \cdot \#\text{Vergleichen}\}$

$n + c + d \cdot \mathbb{E}\{\#\text{Vergleichen}\}$

$n + c + d \cdot \left(2 - \left(\frac{1}{2}\right)^{k-1}\right)$

$$O(n) + \underbrace{O(1) + O(1)}_{\text{Ohne Eingabe } O(1)} = O(n)$$

Aufgabe 3.2

Gegeben sei eine Zahlenfolge $A[0], \ldots, A[n-1]$, 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.

Betrachten wir den folgenden Algorithmus `count(x, A)`:

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

Aufgabe 3.2

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

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).

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

Aufgabe 3.2

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

```
int c = 0;                               → O(1)
for (int i = 0; i < n; i++)              → O(n+1)
    if (A[i] == x) c++;                  → O(n)
return c;                                → O(1)
```

worst-, best- und average-case

$\mathcal{O}(2n+3) \Rightarrow \mathcal{O}(n)$

Aufgabe 3.2

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

Variante 1 (Linear Search):

```
int c = 0;
for (int i = 0; i < n; i++) {
  if (A[i] == x) { c++; }
  else { if (A[i] > x) return c; }
}
return c;
```

$O(n)$ worst-case

$O(1)$ best-case

$O(\frac{n}{2}) + O(1) = O(\frac{n}{2}+1) = O(n)$ average-case

Aufgabe 3.2

Variante 2 (Binary Search):

```
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;
```

$O(1)$ best case

$O(n)$ worst case

$O(\log(n))$ average case

Aufgabe 3.3

Seien $f, g, h : \mathbb{N}_0 \to \mathbb{R}$ Funktionen mit $\exists n_0 : \; \forall n > n_0 : f(n), g(n), h(n) > 0$. Zeigen Sie die folgenden „Transitivitätsregeln“.

(a) $f(n) \in o(g(n)) \quad \Rightarrow \quad f(n) \in \mathcal{O}(g(n))$

(b) $f(n) \in \mathcal{O}(g(n))$ und $g(n) \in \mathcal{O}(h(n)) \quad \Rightarrow \quad f(n) \in \mathcal{O}(h(n))$

(c) $f(n) \in o(g(n))$ und $g(n) \in \mathcal{O}(h(n)) \quad \Rightarrow \quad f(n) \in o(h(n))$

Aufgabe 3.3

(a) $f(n) \in o(g(n))$ $\Rightarrow$ $f(n) \in \mathcal{O}(g(n))$

$f(n) < c \cdot g(n)$

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

Aufgabe 3.3

(b) $f(n) \in \mathcal{O}(g(n))$ und $g(n) \in \mathcal{O}(h(n)) \Rightarrow f(n) \in \mathcal{O}(h(n))$

$$\exists c \quad \exists n_0 \quad \forall n \geq n_0 \quad f(n) \leq c \cdot g(n)$$

$$\exists c' \quad \exists n_0' \quad \forall n \geq n_0' \quad c \cdot g(n) \leq c \cdot c' \cdot h(n)$$

$$f(n) \leq c \cdot g(n) \leq \underbrace{c \cdot c'}_{c''} \cdot h(n)$$

$$n_0'' = \max\{n_0, n_0'\}$$

$$\exists c'' \quad \exists n_0'' \quad \forall n \geq n_0 \quad f(n) \leq c'' \cdot h(n) \qquad f(n) \in \mathcal{O}(h(n))$$

Aufgabe 3.3

(c) $f(n) \in o(g(n))$ und $g(n) \in \mathcal{O}(h(n)) \Rightarrow f(n) \in o(h(n))$

$\forall c > 0 \quad \exists n_0 \quad \forall n \geq n_0 \quad f(n) < c \cdot g(n)$

$\exists c' > 0 \quad \exists n_0' \quad \forall n \geq n_0' \quad g(n) \leq c' \cdot h(n)$

$\exists c' > 0 \quad \forall c > 0 \quad \exists n_0 \, \exists n_0' : \quad f(n) < c \cdot g(n) \wedge g(n) \leq c' \cdot h(n)$

$n_0'' = \max\{n_0, n_0'\}$ $\quad \exists c' > 0 \quad \forall c > 0 \quad \exists n_0'' : \quad f(n) < c \cdot g(n) \wedge c \cdot g(n) \leq c \cdot c' \cdot h(n)$

$c'' = c \cdot c'$ $\quad \underbrace{\forall c'' > 0 \quad \exists n_0'' : \quad f(n) < c'' \cdot h(n)}$ $\quad \Rightarrow \quad f(n) \in o(h(n))$