TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Grundlagen: Algorithmen und Datenstrukturen
Prof. Dr. Helmut Seidl, J. Kranz, A. Reuss, R. Vogler

SS 2016
Übungsblatt 3
01.05.2016

Abgabe: 08.05.2016 (bis 23:59 Uhr)

Aufgabe 3.1 (P) Zufallsvariablen

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. 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[] $(x_1, x_2, \ldots, x_k)$
1 int $i = 1$
2 **while** $i \leq k$ **do**
3 **if** $x_i = a$ **then**
4 **return** *Ja*
5 **end**
6 $i = i + 1$
7 **end**
8 **return** *Nein*

Algorithm 1: IsElementOf

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. $\mathbb{E}[Y] = \mathbb{P}[Y = 1] = c$. 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.

Aufgabe 3.2 (P) Komplexitätsanalyse

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

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

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 die 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 3.3 (P) $\mathcal{O}$h nein, noch mehr Wachstum

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

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

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

Anmerkung: Diese Regeln sind beispielsweise in folgendem Szenario hilfreich: Angenommen, man möchte zeigen, dass $f(n) \in \mathcal{O}(h(n))$ gilt. Wenn f und h sehr komplizierte Funktionen sind, für die ein direkter Beweis schwierig ist, kann man einen Zwischenschritt verwenden: Man formuliert eine Funktion g, sodass $f(n) \in \mathcal{O}(g(n))$ und $g(n) \in \mathcal{O}(h(n))$ gilt, was über die obigen Regeln $f(n) \in \mathcal{O}(h(n))$ impliziert. Hat man g geeignet gewählt (f und g sowie g und h unterscheiden sich nur geringfügig), dann kann der Nachweis von $f(n) \in \mathcal{O}(g(n))$ und $g(n) \in \mathcal{O}(h(n))$ deutlich einfacher sein als der direkte Beweis von $f(n) \in \mathcal{O}(h(n))$. (Natürlich kann man diese Methode auch iterativ anwenden und statt einer einzelnen Zwischenfunktion auch mehrere Zwischenfunktionen $g_1, g_2, \ldots$ verwenden.) Analoge Regeln dieser Art gelten auch für Ω und ω.

Aufgabe 3.4 [4 Punkte] **(H) $\mathcal{O}$hne Ende Wachstum**

Seien $f, g, h : \mathbb{N}_0 \to \mathbb{N}$ Funktionen mit $\exists n_0 \in \mathbb{N} : \forall n > n_0 : f(n), g(n), h(n) > 0$. Welche der folgenden Aussagen treffen zu? Zeigen oder widerlegen Sie jeweils die Aussage!

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

(b) $f(n) \in o(g(n)) \Rightarrow f(n) \notin \Omega(g(n))$

(c) Aus $f(n) \in \Omega(g(n))$ und $h(n) \in o(g(n))$ folgt $f(n) \in \omega(h(n))$

(d) Aus $f(n) \in \mathcal{O}(g(n))$ und $h(n) \in \omega(g(n))$ folgt $f(n) \in o(h(n))$

Aufgabe 3.5 [4 Punkte] **(H) Zufallsvariablen-Caching**

Wir nehmen an, wir haben eine CPU mit einem Cache der Größe c und Arbeitsspeicher der Größe r. Beim Lesen einer Addresse wird zuerst der Cache überprüft. Dieser kann den Wert enthalten (hit) oder nicht (miss). Bei einem miss wird der Wert aus dem Arbeitsspeicher gelesen und dabei in den Cache aufgenommen. Die Zufallsvariable T steht für die Zeit die ein Zugriff in beiden Fällen benötigt:

$$T(hit) = 2ns$$
$$T(miss) = 100ns$$

Sei n die Anzahl der bisher gelesenen unterschiedlichen Addressen. Für die Wahrscheinlichkeit nehmen wir vereinfachend an:

$$Pr[hit] = min(n, c)/r$$
$$Pr[miss] = 1 - Pr[hit]$$

Berechnen Sie $\mathbb{E}[T]$.

Aufgabe 3.6 [12 Punkte] **(H) Dynamisch-doppeltgestapelte Ringschlangen**

In dieser Aufgabe geht es darum, auf Basis eines dynamischen Feldes eine zirkuläre Schlange und einen Stapel zu implementieren. Der Stapel dient außerdem für eine zweite Implementierung einer Schlange. Ein Programmgerüst des in Java zu implementierenden Programms wird zusammen mit diesem Übungsblatt im Ordner `dynamica-ringschlange-angabe/` verteilt. Kommentare im gegebenen Quelltext enthalten wichtige Hinweise und Beispiele; sie sind Teil der Aufgabenstellung.

a) Implementieren Sie zunächst die gekennzeichneten Methoden der Klasse `DynamicArray` . Sie dürfen weitere private Methoden hinzufügen, es dürfen aber keine zusätzlichen statischen Variablen oder Objektvariablen verwendet werden. Achten Sie darauf, dass Sie das dynamische Feld entsprechend der Hinweise weiter unten implementieren. Die Implementierung ist bewusst allgemein gehalten, sodass sie in den folgenden Teilaufgaben verwendet werden kann. Während Ihrer Implementierung sollten Sie auch die Intervall-Klassen vervollständigen.

b) Implementieren Sie nun die gekennzeichneten Methoden der Klasse `DynamicStack`, die einen Stapel auf Basis der Klasse `DynamicArray` umsetzt. Sie dürfen weitere private Methoden hinzufügen, es dürfen aber keine zusätzlichen statischen Variablen oder Objektvariablen verwendet werden.

c) Implementieren Sie die gekennzeichneten Methoden der Klasse `RingQueue`, die eine zirkuläre Warteschlange auf Basis der Klasse `DynamicArray` umsetzt. Sie dürfen weitere private Methoden hinzufügen, es dürfen aber keine zusätzlichen statischen Variablen oder Objektvariablen verwendet werden.

d) Implementieren Sie schließlich die gekennzeichneten Methoden der Klasse `StackyQueue`, die eine Warteschlange auf Basis der Klasse `DynamicStack` umsetzt. Sie dürfen beliebig viele Methoden und Objektvariablen hinzufügen, dabei jedoch keine Java-Klassen außer dem DynamicStack verwenden. Es sind also insbesondere Felder und andere Container verboten.

Beachten Sie die folgenden Hinweise zur Implementierung des dynamischen Feldes:

- Durch das Vergrößern des verwendeten Bereiches ändert sich die Größe des internen Feldes dann und nur dann, wenn das interne Feld das neue Element nicht mehr aufnehmen kann. In diesem Fall gilt für die Länge (*length*) des internen Feldes nach dem Einfügen:

$$length = elemente * growthFactor \tag{1}$$

  Wobei *elemente* die Anzahl verwendeter Elemente im dynamischen Feld und $growthFactor$ den Wachstumsfaktor bezeichnet.

- Durch das Verkleinern des verwendeten Bereiches ändert sich die Größe des internen Feldes dann und nur dann, wenn der bereitgestellte Speicher nach dem Löschvorgang mehr als um das [maximaler Overhead]-fache größer wäre als die Speicheranforderung. In diesem Fall gilt für die Länge (*length*) des internen Feldes ebenfalls nach dem Einfügen:

$$length = elemente * growthFactor \tag{2}$$

  Wobei wieder *elemente* die Anzahl verwendeter Elemente im dynamischen Feld und $growthFactor$ den Wachstumsfaktor bezeichnet.

Bitte beachten Sie, dass eine Fehlerbehandlung nicht verlangt wird; es ist allerdings oft hilfreich, auf Fehlersituationen sinnvoll zu reagieren, da man auf diese Weise Programmierfehler leichter finden kann.

Desweiteren sei nochmals auf unsere Kommunikationsplattform bei Piazza hingewiesen. Dort können Fragen zur Hausaufgabe gestellt und die Fragen anderer Studenten betrachtet und diskutiert werden. In der Vorlesung können Detailfragen zu Hausaufgaben dagegen nur beantwortet werden, wenn Sie uns vor der Vorlesung darauf hinweisen, sodass ein Übungsleiter zur Vorlesung kommen kann.