Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
E. Angriman, F. Nelles, M. Sänger, Dr. P. Schäfer

Berlin, den 24.04.2019

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 2

Abgabe: Mittwoch, den 08.05.2019, bis 11:10 Uhr vor der Vorlesung im Hörsaal. Die Übungsblätter sind in Gruppen von 2/3 Personen zu bearbeiten. Die Lösungen sind auf nach Aufgaben getrennten Blättern abzugeben. Heften Sie bitte die zu einer Aufgabe gehörenden Blätter vor der Abgabe zusammen. Vermerken Sie auf allen Abgaben Ihre Namen, Ihre **CMS-Benutzernamen**, Ihre **Abgabegruppe** (z.B. AG123) aus Moodle, und den **Übungstermin** (z.B. Gruppe 2 oder Mo 13 Uhr bei M.Sänger), an dem Sie Ihre korrigierten Blätter zurückerhalten möchten.
Beachten Sie die Informationen auf der Übungswebseite (https://hu.berlin/algodat19).

Konventionen:

- Mit log wird der Logarithmus $\log_2$ zur Basis 2 bezeichnet.
- Die Menge der natürlichen Zahlen $\mathbb{N}$ enthält die Zahl 0.

Aufgabe 1 (Funktionen ordnen) — $7 \cdot 2 = 14$ Punkte

Beweisen oder widerlegen Sie für die Teilaufgaben (a) bis (g) die folgenden Aussagen:

1. $f \in \mathcal{O}(g)$,
2. $f \in \Omega(g)$.

Hinweis: Benutzen Sie entweder die Definitionen direkt, oder betrachten Sie den Grenzwert des Quotienten der Funktionen und wenden Sie falls nötig den Satz von l'Hôpital an.

	$f(n)$	$g(n)$
(a)	$n+1$	$n+1000$
(b)	$3n^2$	n^2+1000
(c)	2^n	$n2^n$
(d)	2^n	2^{2n}
(e)	n	$\log(n)$
(f)	$\sqrt{n}$	$(\log(n))^2$
(g)	$\sqrt{\log n}$	$\log\sqrt{n}$

Aufgabe 2 (Eigenschaften der $\mathcal{O}$-Notation) **(2+2+2)+4+4 = 14 Punkte**

Beweisen Sie die folgenden Aussagen.

1. Seien f, g und h Funktionen, die von $\mathbb{N}$ nach $\mathbb{R}_{\geq 0}$ abbilden. Sei $f_0 \in \mathcal{O}(f)$ und $g_0 \in \mathcal{O}(g)$. Dann gilt:
   a) $f_0 \cdot g_0 \in \mathcal{O}(f \cdot g)$.
   b) $f \notin o(f)$.
   c) $f \in \Omega(g)$ und $g \in \Omega(h) \quad \Longrightarrow \quad f \in \Omega(h)$.
2. Sei $p\colon \mathbb{N} \to \mathbb{R}$ mit $p(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0$ ein Polynom in n vom Grad d mit $a_i \in \mathbb{R}_{\geq 0}$ für $0 \leq i \leq d$ und $a_d > 0$. Dann gilt $p(n) \in \Theta(n^d)$.
3. Seien $a, b \in \mathbb{R}$ positive Konstanten, mit b > 0. Dann gilt $(n + a)^b \in \Theta(n^b)$.

Aufgabe 3 (Stacks und Queues) **6 + 6 = 12 Punkte**

In dieser Aufgabe sollen Sie zwei Algorithmen in Pseudocode entwerfen, die mit Stacks und Queues arbeiten. Bei der Verwendung eines Stacks oder einer Queue stehen Ihnen nur die für Stacks und Queues in der Vorlesung kennengelernten Methoden (also `enqueue()`, `dequeue()`, `head()` und `isEmpty()` für Queues bzw. `push()`, `pop()`, `top()` und `isEmpty()` für Stacks) zur Verfügung. Es existiert insbesondere keine Funktion zur Bestimmung der Länge eines Stacks/einer Queue.
Ein Palindrom ist eine Zeichenkette, die von vorne und von hinten gelesen identisch ist. Beispiele hierfür sind die Zeichenketten

- „anna“,
- „rentner“,
- „erikafeuertnuruntreuefakire“ oder
- „tsssvssst“.

Anagramme sind Zeichenketten, die durch Umstellung der Buchstaben (Permutation) ineinander umgewandelt werden können. Beispiele hierfür sind die Zeichenketten

- „regal“, „lager“ und „aeglr“,
- „angstbude“, „bundestag“, „dantesbug“ und „abdegnstu“ oder
- „wolfgangamadeusmozart“ und „afamousgermanwaltzgod“.

1. Entwerfen Sie einen Algorithmus in Pseudocode, der als Eingabe eine Zeichenkette der Länge n über dem Alphabet $\Sigma = \{a, b, \ldots, z\}$ in Form einer Queue erhält und in Laufzeit $\mathcal{O}(n)$ testet, ob es sich bei dieser Zeichenkette um ein Palindrom handelt. Als Eingabe erhält der Algorithmus ausschließlich eine Queue q von Buchstaben, d.h., Sie haben zunächst nur Zugriff auf den ersten Buchstaben der Zeichenkette. Abgesehen von der Eingabe q, die beliebig modifiziert werden kann, darf Ihr Algorithmus lediglich Variablen von elementaren Datentypen (und kein Array) sowie entweder einen zusätzlichen Stack oder eine zusätzliche Queue verwenden. Notieren Sie Ihren Algorithmus in Pseudocode und begründen Sie, warum seine Laufzeit in $\mathcal{O}(n)$ liegt.

2. Entwerfen Sie einen Algorithmus in Pseudocode, der als Eingabe eine Zeichenkette der Länge n über dem Alphabet $\Sigma = \{a, b, \ldots, z\}$ in Form einer Queue erhält und in Laufzeit $\mathcal{O}(n)$ testet, ob diese Zeichenkette ein Anagramm eines Palindroms ist, d.h., ob sich aus dieser Zeichenkette durch Permutation der Buchstaben irgendein Palindrom erzeugen lässt. Dieses Palindrom muss kein sinnvolles Wort sein. Ihr Algorithmus muss das Palindrom auch nicht ausgeben, es reicht **true** oder **false** als Ausgabe. Auch hier erhält der Algorithmus als Eingabe eine Queue q von Buchstaben. Abgesehen von der Eingabe q, die beliebig modifiziert werden kann, darf Ihr Algorithmus lediglich Variablen von elementaren Datentypen (und kein Array) sowie entweder einen zusätzlichen Stack oder eine zusätzliche Queue verwenden. Notieren Sie Ihren Algorithmus in Pseudocode und begründen Sie, warum seine Laufzeit in $\mathcal{O}(n)$ liegt.

Aufgabe 4 (Stacks) **12 Punkte**

In dieser Aufgabe geht es um das Auswerten von arithmetischen Ausdrücken, welche ausschließlich aus Ziffern, öffnenden und schließenden Klammern und den Operatoren $+$, $-$, $*$, / bestehen. Hierbei steht der Operator „/“ für Ganzzahldivision.
Wir definieren einen *wohlgeformten arithmetischen Ausdruck* wie folgt:

- Eine Ziffer $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist ein wohlgeformter arithmetischer Ausdruck.
- Sind X und Y wohlgeformte arithmetische Ausdrücke, dann sind auch die vier Ausdrücke $(X + Y)$, $(X - Y)$, $(X * Y)$ und (X/Y) wohlgeformte arithmetische Ausdrücke.

Beispiele für wohlgeformte arithmetische Ausdrücke entsprechend dieser Definition sind unter anderem: „$((8 + 7) * 2)$“, „$(4 - (7 - 1))$“ oder „8“. Bei den Ausdrücken „$)8+)1(())$“, „$(8 + ())$“, „-1“, „$(\quad 5 \quad - 7)$“, „108“ oder „(8)“ handelt es sich hingegen nicht um wohlgeformte arithmetische Ausdrücke.

Schreiben Sie ein Java-Programm, welches für eine übergebene Zeichenkette überprüft, ob es sich um einen wohlgeformten arithmetischen Ausdruck handelt. Falls es sich um einen wohlgeformten arithmetischen Ausdruck handelt, so soll Ihr Algorithmus das Ergebnis dieses Ausdrucks ausrechnen und als Integer-Wert zurückgeben. Andernfalls soll eine ExpressionNotWellFormedException geworfen werden. Für diese Aufgabe bietet sich die Verwendung eines Stacks (java.util.Stack) an. Zur Vereinfachung der Implementierung, brauchen Sie beim Ausrechnen des Ergebnisses Divisionen durch 0 sowie arithmetische Überläufe nicht gesondert behandeln.

Ergänzen Sie dazu den fehlenden Code in der Vorlage `Parser.java`, welche Sie auf der Übungswebseite[1] vorfinden. Sie können beliebige neue Variablen und Hilfsmethoden zur Klasse hinzufügen, dürfen jedoch keine außer den von Java bereitgestellten Standard-Bibliotheken verwenden. Stellen Sie sicher, dass alle Testfälle in der main-Methode der Datei Parser.java erfolgreich durchlaufen. Achten Sie außerdem auf Randbedingungen und Spezialfälle, die von den Testfällen vielleicht nicht vollständig abgedeckt werden.

Hinweis zur Abgabe: Ihr Java-Programm muss auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, sodass die einzelnen Schritte nachvollziehbar sind. Die Abgabe des von Ihnen modifizierten Source-Codes `Parser.java` erfolgt über Moodle.

[1] https://hu.berlin/algodat19