Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Marius Kloft
Bux, Grußien, Schäfer, Tschorsch, Völlinger

Berlin, den 02.05.2016

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 2

Abgabe: Mittwoch den 18.5.2016 bis 10:55 Uhr vor der Vorlesung im Hörsaal oder bis 10:45 Uhr im Briefkasten (RUD 25, Raum 3.321). Die Übungsblätter sind in Gruppen von zwei (in Ausnahmefällen auch drei) 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 Matrikelnummern, den Namen Ihrer Goya-Gruppe und an welchem Übungstermin (Wochentag, Uhrzeit, Dozent) Sie die korrigierten Abgaben abholen möchten. Beachten Sie auch die aktuellen Hinweise auf der Übungswebsite unter https://hu.berlin/algodat16.

Konventionen:

- Mit der Aufforderung „analysieren Sie die Laufzeit“ ist gemeint, dass Sie eine möglichst gute obere Schranke der Zeitkomplexität angeben sollen und diese begründen sollen.

Aufgabe 1 (Operationen auf Stacks) — (2 + 3 + 3) + 5 = 13 Punkte

a) In der Vorlesung haben Sie bereits die Datenstruktur Stack kennengelernt. Stacks stellen üblicherweise die folgenden Funktionen zur Verfügung:

- `push(value)`: Legt den Wert `value` oben auf dem Stack ab.
- `pop()`: Entfernt den obersten Wert vom Stack und gibt ihn zurück. Wenn der Stack leer ist, wird der Wert `null` zurückgegeben.
- `top()`: Liefert den obersten Wert vom Stack ohne ihn dabei zu entfernen. Wenn der Stack leer ist, wird der Wert `null` zurückgegeben.
- `isEmpty()`: Gibt `true` zurück, falls der Stack keine Werte enthält und `false` andernfalls.

Implementieren Sie die folgenden Funktionen für einen Stack zum Speichern von Integer-Werten in Java:

- `dup()`: Dupliziert den obersten Integer-Wert auf dem Stack, falls vorhanden.
- `size()`: Liefert die Größe des Stacks, d.h. die Anzahl der Werte im Stack.
- `reverse()`: Kehrt den Stack um, d.h. vertauscht die Reihenfolge aller Werte.

Ergänzen Sie den fehlenden Code in der Vorlage `IntStack.java`, welche Sie auf der Website vorfinden, unter Ausnutzung der existierenden Funktionen-Stubs. Sie benötigen zum Kompilieren auch die Datei `Element.java`. Zum Testen können Sie die `main`-Methode in der Vorlage `IntStack.java` verwenden. Achten Sie insbesondere auf Randbedingungen und Spezialfälle.

Hinweis zur Abgabe: Ihr Java-Programm muss unter *Java 1.7* auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, so dass die einzelnen Schritte nachvollziehbar sind. Die Abgabe erfolgt über Goya.

b) Wie würden Sie die in der Vorlage `IntStack.java` gegebenen Datenstrukturen und Funktionen (ausgenommen die von Ihnen hinzugefügten Funktionen) anpassen, um eine Funktion `getMax()` zu implementieren, die den größten Wert des aktuellen Stacks zurück gibt? (Falls notwendig, dürfen Sie auch Datenstrukturen und Funktionen in `Element.java` ändern.) Die Funktionen `pop()`, `push(value)` und `getMax()` sollen dabei allesamt eine Laufzeit von $\mathcal{O}(1)$ aufweisen. Sie dürfen jedoch beliebig viel zusätzlichen Speicherplatz verwenden. Es ist ausreichend, wenn Sie ihren Lösungsansatz skizzieren und kein lauffähiges Programm schreiben. Erläutern Sie welche Änderungen notwendig wären.

Hinweis zur Abgabe: Geben Sie die Lösung dieser Teilaufgabe schriftlich ab.

Aufgabe 2 (Stacks und Queues) — 9 + 4 = 13 Punkte

Es soll eine Queue Q unter Verwendung von zwei Stacks $S1$, $S2$ mit den Funktionen `push(value)`, `pop()`, `top()` und `isEmpty()` aus Aufgabe 1 und keiner sonstigen Datenstruktur implementiert werden.

a) Entwerfen Sie Algorithmen für die folgenden vier Funktionen einer Datenstruktur Queue (nur unter Verwendung zweier Stacks mit den vier o.g. Methoden, d.h., es darf keine weitere interne Listenstruktur verwendet werden). Die Algorithmen sollten insgesamt möglichst effizient sein. Notieren Sie Ihre Algorithmen als Pseudocode.

- `enqueue(value)`: Hängt den Wert `value` an das Ende der Queue an.
- `dequeue()`: Entfernt den ersten Wert vom Anfang der Queue und gibt ihn zurück. Wenn die Queue leer ist, wird der Wert `null` zurückgegeben.
- `head()`: Liefert den ersten Wert vom Anfang der Queue zurück ohne ihn dabei zu entfernen. Wenn die Queue leer ist, wird der Wert `null` zurückgegeben.
- `isEmpty()`: Gibt `true` zurück, falls die Queue keine Werte enthält und `false` andernfalls.

b) Analysieren Sie die Laufzeit für die vier genannten Funktionen in Abhängigkeit der Größe der Queue, d.h. der Anzahl n ihrer Werte. Nehmen Sie dabei an, dass alle Funktionen der Stacks in $\mathcal{O}(1)$ laufen.

Aufgabe 3 (Queues und Glückszahlen) — 2 + (6 + 4) = 12 Punkte

a) Es seien zwei Queues $Q1$ und $Q2$ gegeben, die jeweils beliebige natürliche Zahlen sortiert in aufsteigender Reihenfolge (ohne Duplikate) enthalten. ($Q1$`.head()` bzw. $Q2$`.head()` liefert die kleinste Zahl in $Q1$ bzw. $Q2$). Erläutern Sie, wie Sie genau die Zahlen, die sowohl in $Q1$ als auch $Q2$ enthalten sind, in aufsteigender Reihenfolge ausgeben können, ohne dabei weitere Datenstrukturen zu verwenden. (Es ist nicht notwendig und nicht ausreichend hier Pseudocode anzugeben.)

b) *Glückszahlen* u haben die folgende Form: $u = 2^x \cdot 3^y \cdot 5^z$, mit $x, y, z \geq 0$. D.h. 1 ist die kleinste Glückszahl und für jeden Primfaktor p einer Glückszahl $u > 1$ gilt: $p \in \{2, 3, 5\}$.

1. Entwerfen Sie einen Algorithmus, der bei Eingabe von k in Zeit $\mathcal{O}(k)$ die ersten k Glückszahlen in aufsteigender Reihenfolge ausgibt. Bei der Eingabe von 10 soll Ihr Algorithmus zum Beispiel die Zahlen $1, 2, 3, 4, 5, 6, 8, 9, 10, 12$ ausgeben. Benutzen Sie dazu in geeigneter Weise drei Queues. Nehmen Sie dabei an, dass die Funktionen

`head()`, `enqueue(element)`, `dequeue()`, `isEmpty()` der Queues in $\mathcal{O}(1)$ laufen. Notieren Sie Ihren Algorithmus in Pseudocode und analysieren Sie die Laufzeit Ihres Algorithmus.

Hinweis: Es ist nicht möglich, generierte Glückszahlen zu sortieren, da sonst die Laufzeitschranke verletzt wird.

2. Beweisen Sie die Korrektheit Ihres Algorithmus.

Aufgabe 4 (Infix und Postfix) **4 + 4 + 4 = 12 Punkte**

Infix-Ausdrücke sind wie folgt rekursiv definiert: Eine Ziffer $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist ein Infix-Ausdruck, und für alle Infix-Ausdrücke a_1 und a_2 sind $(a_1 + a_2)$, $(a_1 - a_2)$, $(a_1 * a_2)$ und (a_1/a_2) Infix-Ausdrücke.

Ähnlich sind *Postfix-Ausdrücke* rekursiv definiert: Eine Ziffer $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist ein Postfix-Ausdruck, und für alle Postfix-Ausdrücke a_1 und a_2 sind $a_1 a_2 +$, $a_1 a_2 -$, $a_1 a_2 *$ und $a_1 a_2 /$ Postfix-Ausdrücke.

Semantisch entspricht der Infix-Ausdruck $(3 - (7 * 2))$ dem Postfix-Ausdruck $3\ 7\ 2 * -$ bzw. der Infix-Ausdruck $((7 + 1) * ((3 - 6) * (5 - 2)))$ dem Postfix-Ausdruck $7\ 1 + 3\ 6 - 5\ 2 - **$.

Ergänzen Sie den fehlenden Code in der Vorlage `Postfix.java`, welche Sie auf der Website vorfinden, unter Ausnutzung der existierenden Funktionen-Stubs. Zum Testen können Sie die `main`-Methode in der Vorlage `Postfix.java` verwenden.

a) Implementieren Sie die Funktion `evalPostfix(postfix)`, die einen arithmetischen Ausdruck in Postfix-Notation auswertet.

b) Implementieren Sie die Funktion `infixToPostfix(infix)`, die einen arithmetischen Ausdruck von Infix-Notation nach Postfix-Notation konvertiert.

c) Implementieren Sie die Funktion `postfixToInfix(postfix)`, die einen arithmetischen Ausdruck von Postfix-Notation nach Infix-Notation konvertiert.

Sie können davon ausgehen, dass es sich bei den Ausdrücken, die den von Ihnen implementierten Funktionen übergeben werden, um gültige Infix- bzw. Postfix-Ausdrücke handelt.

Hinweis: Benutzen Sie für diese Aufgabe die Klasse `java.util.Stack`.

Hinweis zur Abgabe: Ihr Java-Programm muss unter *Java 1.7* auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, so dass die einzelnen Schritte nachvollziehbar sind. Die Abgabe erfolgt über Goya.