Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Stefan Kratsch
Dr. K. Ahrens, F. Hegerfeld, F. Nelles, Dr. P. Schäfer

Berlin, den 30.04.2018

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 2

Abgabe: Montag den 14.05.2018 bis 11:10 Uhr vor der Vorlesung im Hörsaal oder bis 10:45 Uhr im Briefkasten (RUD 25, Raum 3.321). Die Übungsblätter sind in Gruppen von 2 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 Ihren Übungstermin (z.B. Do 13 Uhr bei Florian Nelles), zu dem Sie Ihre korrigierten Blätter zurückerhalten werden.
Beachten Sie die Informationen auf der Übungswebseite (`https://hu.berlin/algodat18`).

Konventionen:

- Für ein Array A ist $|A|$ die Länge von A, also die Anzahl der Elemente in A. Die Indizierung aller Arrays auf diesem Blatt beginnt bei 1 (und endet also bei $|A|$).
- 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

1. 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 der Stack nicht leer ist, andernfalls wird keine Änderung am Stack vorgenommen.
   - `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 1.8* auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, so dass die einzelnen Schritte nachvollziehbar sind. Die Abgabe erfolgt über Moodle.

2. Geben Sie die Lösung dieser Teilaufgabe schriftlich auf einem Blatt ab: Wie würden Sie die 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? 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.

Aufgabe 2 (Algorithmenentwurf) **9 + 3 = 12 Punkte**

Für eine Sequenz von n Zahlen $z_1, \ldots, z_n$ ist der Median allgemein definiert als der Wert, der an mittlerer Stelle steht, wenn man die Werte der Größe nach sortiert. Für eine aufsteigend sortierte Sequenz $z_1, \ldots, z_n$ mit $z_1 \leq \cdots \leq z_n$ ist der Median das Element $z_{(n+1)/2}$, falls n ungerade und das Element $z_{n/2+1}$, falls n gerade. Zum Beispiel: der Median der Sequenz $1, 10, 11$ ist 10 und der Median der Sequenz $0, 1, 2, 10$ ist 2.
Gegeben seien nun zwei aufsteigend sortierte Arrays $A = [a_1, \ldots, a_n]$ und $B = [b_1, \ldots, b_n]$, mit jeweils n Elementen. Der Einfachheit halber wollen wir annehmen, dass alle $2n$ Elemente verschieden sind.

1. Entwerfen Sie einen möglichst effizienten Algorithmus der den Median der Elemente beider Arrays bestimmt. Unter dem Median der Elemente beider Arrays ist der Median der (unsortierten) Sequenz $a_1, \ldots, a_n, b_1, \ldots, b_n$ zu verstehen.

   Hinweis: Für einen korrekten sublinearen Algorithmus gibt es 9 Punkte, ein linearer Algorithmus gibt 6 Punkte, eine superlineare Variante 3 Punkte.

2. Analysieren Sie die Laufzeit Ihres in Pseudocode vorgestellten Algorithmus.

Aufgabe 3 (Queues und Glückszahlen) **(6 + 3 + 4) = 13 Punkte**

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

   Hinweis: Es ist nicht ausreichend, generierte Glückszahlen mit einem allgemeinen Sortierverfahren in $\mathcal{O}(n \log n)$ zu sortieren, da sonst die Laufzeitschranke verletzt wird.

2. Analysieren Sie die Laufzeit Ihres in Pseudocode vorgestellten Algorithmus.

3. Beweisen Sie die Korrektheit Ihres Algorithmus durch vollständige Induktion.

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.

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

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

3. 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 1.8* auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, so dass die einzelnen Schritte nachvollziehbar sind. Die Abgabe erfolgt über Moodle.