Aufgabenblatt 5
Grundlegende Datenstrukturen

- Abgabetermin: **Freitag, 27.06.14 23:59 Uhr**
- Zur Prüfungszulassung muss ein Aufgabenblatt mit mind. 25% der Punkte bewertet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Abgabe:
  - Geben Sie im Abgabesystem[1] eine ZIP-Datei ab, welche den Quelltext aller implementierten Klassen (Java 7), sowie entsprechende textuelle Lösungen enthält.
  - Achten Sie darauf, dass ihr ZIP-Archiv *keine* kompilierten Klassen (class-Dateien) enthält.
  - Textuelle Lösungen reichen Sie bitte in einer entsprechend benannten *PDF*-Datei (aufgabe_x.pdf) auf oberster Ebene im ZIP-Archiv ein.
  - Der Quelltext soll in einer Ordnerstruktur analog zu assignment5_sources.zip[2] vorliegen: Quellcode im `src` Ordner, etc.
  - Achten Sie darauf, dass Ihr Quelltext gut kommentiert ist.
  - Geben Sie bei der Einreichung Ihrer Lösung den Namen des anderen Gruppenmitglieds unter "Authors" an (pro Gruppe nur eine Abgabe).

Vorbereitung

Laden Sie sich das Archiv assignment5_sources.zip[2] herunter, es enthält das Gerüst (Klassen- sowie Methodendeklarationen und UnitTests), auf dem die folgenden Aufgaben aufbauen.

[1] `https://www.dcl.hpi.uni-potsdam.de/submit/`
[2] `http://hpi-web.de/fileadmin/hpi/FG_Naumann/lehre/SS2014/PTII/assignment5_sources.zip`

Aufgabe 1: Sequenzen

9 P

Implementieren Sie folgende Klassen zur effizienten Speicherung von Datensequenzen (dynamischer Länge), die Elemente eines generischen Typs [3] (`T`) beinhalten[4]:

`assignment5.ArraySequence<T>` soll Daten in Form eines Arrays speichern
`assignment5.SinglyLinkedSequence<T>` soll Daten als einfach verkettete Liste speichert
`assignment5.DoublyLinkedSequence<T>` soll Daten als doppelt verkettete Liste speichert

Stellen Sie sicher, dass alle drei Klassen die folgenden Interfaces/abstrakten Datentypen (ADT) aus dem package `assignment5.adt` implementieren:

`Stack<T>` – ein Stapelspeicher, bei dem Elemente übereinander gestapelt und in umgekehrter Reihenfolge vom Stapel entnommen werden.
`Queue<T>` – eine Warteschlange, die eine Menge von Objekten aufnehmen kann und diese in der Reihenfolge ihres Einfügens zurückgibt.
`List<T>` – eine Sequenz mit wahlfreiem Zugriff.

Achten Sie darauf, dass sich die Datenstrukturen ggf. vergrößern müssen um mehr Elemente speichern zu können. Sollte eine Operation ausgeführt werden, die nicht möglich ist (z.B. Entnehmen aus einer leereren Sequenz) soll die entsprechende Methode eine `IllegalStateException` werfen. Wie immer gilt: UnitTests helfen beim Finden von Fehlern.

Implementieren Sie anschließend noch eine von `assignment5.ArraySequence<T>` abgeleitete Klasse **`assignment5.CircularBuffer<T>`**, die das Wirkprinzip eines Ringpuffers besitzt[5]. Die maximale Größe des Puffers soll im Konstruktor übergeben werden und im Anschluss nicht änderbar sein. Die Methoden für den wahlfreien Zugriff (`List<T>`) sollen sich wie folgt verhalten:

```
CircularBuffer<Character> buf = new CircularBuffer<>(2);
buf.insert(0, '0'); buf.insert(1, '1'); buf.insert(1, '2');
buf.get(0); // -> '0'
buf.get(1); // -> '2'
buf.get(2); // -> Exception
buf.insert(2, '3'); // -> Exception
buf.change(0, '4');
buf.get(0); // -> '4'
buf.get(1); // -> '2'
buf.remove(0);
buf.get(0); // -> '2'
buf.get(1); // -> Exception
buf.length(); // -> 1
```

Gestalten Sie die Funktionsweise der Methoden von `Stack<T>` und `Queue<T>` analog zur Funktionsweise des Ringpuffers[5]. Achten Sie bei ihrer Implementierung darauf, möglichst viele Funktionalitäten von der Elternklasse zu übernehmen.

Vergleichen Sie die Komplexität ihrer Implementierung aller durch die abstrakten Datentypen geerbten Methoden. Erstellen Sie dazu ähnlich zu Tabelle 1 eine Übersicht über die Laufzeitkomplexität der Methoden, wobei n der Länge/dem Füllstand der Sequenzen entspricht. Speichern Sie die Übersicht in einer Datei namens aufgabe_1.pdf ab und fügen Sie sie dem Abgabearchiv hinzu.

Tabelle 1: Beispieltabelle (keine Garantie auf Korrektheit).

	`ArraySeq.`	`SinglyLinkedSeq.`	`DoublyLinkedSeq.`	`CircularBuffer`
`push`	$\mathcal{O}(n!)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$	$\mathcal{O}(n^3)$
`pop`	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^5)$	$\mathcal{O}(\log n)$
`...`				

[3] `http://docs.oracle.com/javase/tutorial/extra/generics/simple.html`
[4] ohne Verwendung der Collection-API Klassen
[5] `http://en.wikipedia.org/wiki/Circular_buffer`

Aufgabe 2: Bäume

6 P

Implementieren Sie die Klasse `assignment5.BinarySearchTree<T>`, die zum Speichern von Daten einen binären Suchbaum (BST) aufbaut und vom ADT `assignment5.adt.Set` abgeleitet ist[6]. Die Datenstruktur darf explizit *nicht* vor Entartung gefeit sein (unbalanciert).

Ermitteln Sie nun die durchschnittliche Suchpfadlänge in einem zufällig erzeugten BST. Gehen sie dabei wie folgt vor:

1. Erstellen sie eine zufällige Element-Sequenz s der Länge $n = 2^x - 1, x \in \mathbb{N}^+$ (Länge n nicht zu klein wählen, z.B. $x = 10$)
2. Lassen Sie mit dieser Sequenz eine leere Instanz der Klasse `BinarySearchTree` füllen (Einfügereihenfolge analog zur zufälligen Sequenz)
3. Ermitteln Sie über alle Elemente der Sequenz s die durchschnittliche Suchpfadlänge $\bar{l}$ im entstandenen Baum (der Suchpfad zum Wurzelelement i besitzt dabei die Suchpfadlänge $l(i) = 1$, der Suchpfad eines Knotens j aus Ebene 1 besitzt die Länge $l(j) = 2$ etc.)
4. Wiederholen Sie das Experiment (Schritt 1-3) k-mal und speichern Sie sich alle durchschnittlichen Suchpfadlänge $\overline{l_1}, \ldots, \overline{l_k}$. Für alle Wiederholungen muss n gleich bleiben. Achten sie darauf, dass Sie k nicht zu klein wählen.

Beschreiben Sie ihr Setup (z.B. Größe für n, Wiederholungen $k, \ldots$) kurz und stellen Sie die Ergebnisse für Durchläufe ($\overline{l_i}$) in einem Histogramm[7] dar. Die durchschnittlichen Suchpfadlängen über mehrere Wiederholungen sind normalverteilt. Deshalb können Sie den Erwartungswert $\mathbb{E}(\overline{l_{unbal}})$ der durchschnittliche Suchpfadlänge für zufällig erzeugte unbalancierte BST als Mittelwert über ihre k Wiederholungen schätzen:

$$\mathbb{E}(\overline{l_{unbal}}) \approx \frac{1}{k} \sum_{i=1}^{k} \overline{l_i}$$

Vergleichen Sie diesen Wert mit der zu erwartenden durchschnittlichen Suchpfadlänge in einem vollständig balancierten BST ($h = \log_2(n+1)$):

$$\mathbb{E}(\overline{l_{bal}}) = \frac{1}{n} \sum_{i=0}^{h-1} (i+1) \cdot 2^i = \log_2(n+1) + \frac{\log_2(n+1)}{n} - 1$$

Diskutieren Sie ihre Beobachtungen kurz und speichern Sie die Diskussion sowie Histogramm und Setup in einer PDF-Datei aufgabe_2.pdf ab und fügen Sie sie dem Abgabearchiv hinzu.

[6] ohne Verwendung der Collection-API Klassen
[7] `http://de.wikipedia.org/wiki/Histogramm`

Aufgabe 3: Heaps

5 P

a) Implementieren Sie die Klassen `assignment5.MinHeapArray` und `assignment5.MinHeapTree`, die ihre Daten in Form eines Arrays bzw. eines Baumes ablegen und das Interface `assignment5.adt.MinHeap` (binärer Min-Heap) implementieren[8].

b) Diskutieren sie kurz die Unterschiede der beiden Implementierungen im Speicherverbrauch in einer PDF-Datei aufgabe_3.pdf und fügen Sie sie dem Abgabearchiv hinzu.

c) Implementieren sie die Methode `assignment5.Sort#heapsort(T[] a)` unter Verwendung der obigen Datenstrukturen.

[8] ohne Verwendung der Collection-API Klassen

Zusatzaufgabe: Maulwurfsbau

Der Bau des kleinen Maulwurfs besteht aus N Kammern und M Gängen. Jeder Gang verbindet genau zwei Kammern, und es gibt *höchstens* eine Möglichkeit, sich unterirdisch zwischen jedem Paar von Kammern zu bewegen. Einige der Kammern haben einen direkten Zugang zu einem Erdhügel (und somit zu einem Ausgang), wobei jeder Erdhügel nur mit genau einer Kammer direkt verbunden ist. Wenn zwei Kammern einen direkten Zugang zu einem Erdhügel besitzen, weht unterirdisch immer eine leichte Brise zwischen diesen Kammern (und damit auch in allen Kammern dazwischen).

Nun interessiert den kleinen Maulwurf, wieviele Kammern belüftet sind. Außerdem möchte er wissen, wie groß die Zahl der Paare von Zimmern ist, zwischen denen ein Lüftchen weht.

Implementieren Sie eine effiziente Methode `connections(boolean[] cellHasHill, int[][] passages)` der Klasse `assignment5.Molehill`, die die zwei gesuchten Werte in Form eines Arrays zurückgibt (1. Wert: Anzahl der belüfteten Kammern; 2. Wert: Kammerpaare mit bestehendem Luftzug).
Dabei entpricht `boolean[] cellHasHill` einem Array, das für jede Kammer angibt, ob sie mit einem Maulwurfshügel verbunden ist.Aus der Belegung `cellHasHill = boolean[] {true, false, true}` ergeben sich zwei Maulwurfshügel, eine über Kammer null und eine über Kammer zwei.
Weiterhin entspricht `int[][] passages` einem Array, das alle Gänge beinhaltet. Aus der Belegung `passages = new int[][] {{1,0},{0,2}}` ergeben Sich also 2 Gänge: einer zwischen Kammer 0 und 1 sowie einer zwischen Kammer 0 und 2. Außerdem ist $N =$ `cellHasHill.length` sowie $M =$ `passages.length`.

Wertebereiche
$1 \leq N \leq 50\ 000$
$0 \leq M \leq N - 1$

Beispiel

```
boolean[] cellHasHill = boolean[] {true,true,true,true,true,false};
int[][] passages = new int[][] {{0,1},{0,5},{0,4},{1,3},{3,2}};
connections(cellHasHill, passages); // -> {5,10}
```