Universität Stuttgart

Institut für Visualisierung
und Interaktive Systeme

Übungsblatt 1

Datenstrukturen und Algorithmen (SS 2017)

Abgabe: Mittwoch, 26.04.2017, 23:59 Uhr — Besprechung: ab Montag, 08.05.2017

Bitte lösen Sie die Übungsaufgabe in **Gruppen von 3 Studenten** und wählen EINEN Studenten aus, welcher die Lösung im ILIAS als **PDF** als **Gruppenabgabe** (unter Angabe aller Gruppenmitglieder) einstellt. Bitte erstellen Sie dazu ein **Titelblatt**, welches die Namen der Studenten, die Matrikelnummern, und die E-Mail-Adressen enthält.

Die Aufgaben mit Implementierung sind mit Impl gekennzeichnet. Das entsprechende Eclipse-Projekt kann im ILIAS heruntergeladen werden. Bitte beachten Sie die Hinweise zu den Implementierungsaufgaben, die im ILIAS verfügbar sind.[1]

Dieses Übungsblatt beinhaltet 4 Aufgaben mit einer Gesamtzahl von 30 Punkten.

Aufgabe 1 Suchverfahren *[Punkte: 7]*

Illustrieren Sie gemäß des in der Vorlesung vorgestellten Schemas (inklusive Nummerierung der einzelnen Schritte) jeweils die *sequenzielle* und die *binäre* Suche für die unten gezeigten Folgen, oder begründen Sie, wieso sich ein Suchverfahren nicht anwenden lässt.

(a) *(4 Punkte)* Gesuchtes Element: **53**

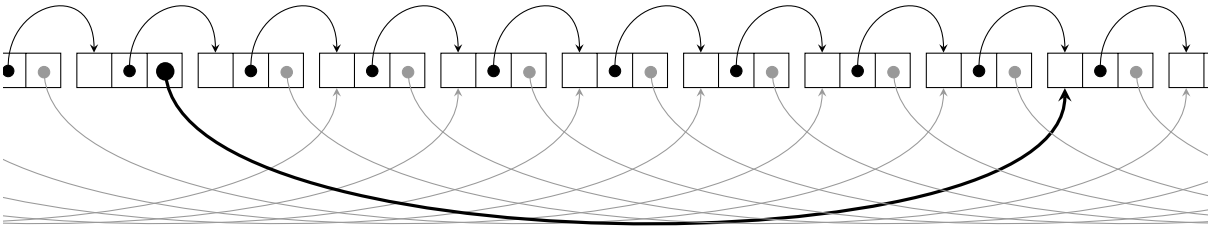
18	21	34	36	46	53	55	57	62	73	76	81	85	88	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

(b) *(3 Punkte)* Gesuchtes Element: **64**

12	15	21	22	24	31	40	48	55	59	71	88	64	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Aufgabe 2 Impl Verkettete Listen *[Punkte: 10]*

Im Folgenden ist eine verkettete Liste dargestellt. Zusätzlich zu der Verkettung mit direkten Nachbarn ist jeder Knoten auch noch mit – soweit vorhanden – seinem achtnächsten Nachbarn verbunden (zur Verdeutlichung ist eine der Abkürzungskanten exemplarisch hervorgehoben):



Implementieren Sie die oben abgebildete Listenstruktur, bestehend aus einer Klasse `SpeedList<T>` (die eine Schnittstelle `ISpeedList` implementiert) und einer dazugehörigen Knotenklasse. **Die Verwendung von java.util.LinkedList etc. ist dabei nicht erlaubt, da Sie die Implementierung selbst vornehmen sollen!** In ihrer Implementierung sollen, wann immer möglich und sinnvoll,

[1] `https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_fold_1214489.html`

die Abkürzungskanten verwendet werden, um die Navigation innerhalb der Liste zu beschleunigen. Die Schnittstelle sowie die zu implementierende Klasse sind im Eclipse-Projekt zu dieser Aufgabe enthalten.

Aufgabe 3 Impl Stacks und Queues *[Punkte: 9]*

In der Vorlesung haben Sie die Datenstrukturen Stack (LIFO) und Queue (FIFO) kennengelernt. Im Eclipse-Projekt zu dieser Aufgabe sind die zwei Schnittstellen für diese Datenstrukturen gegeben (`IStack` und `IQueue`, siehe Listings 1 und 2). Implementieren Sie die Klassen `Stack` und `Queue`. **Die Verwendung von java.util.Stack und java.util.Queue ist dabei nicht erlaubt, da Sie die Implementierung selbst vornehmen sollen!**

Listing 1: IStack.java

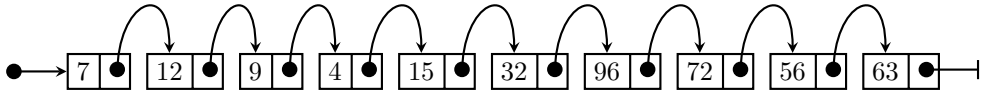
```
1  package de.unistuttgart.dsass2017.ex01.p3;
2
3  public interface IStack<T> {
4
5          /** Adds new element to the top */
6          public void push(T t);
7          /** Removes and returns the top element */
8          public T pop();
9          /** Returns the top element without removing */
10         public T top();
11         /** Checks if the stack is empty */
12         public boolean isEmpty();
13
14 }
```

Listing 2: IQueue.java

```
1  package de.unistuttgart.dsass2017.ex01.p3;
2
3  public interface IQueue<T> {
4
5          /** Enqueues an element */
6          public void enqueue(T t);
7          /** Dequeues the first element */
8          public T dequeue();
9          /** Returns the first element */
10         public T front();
11         /** Checks if the queue is empty */
12         public boolean isEmpty();
13
14 }
```

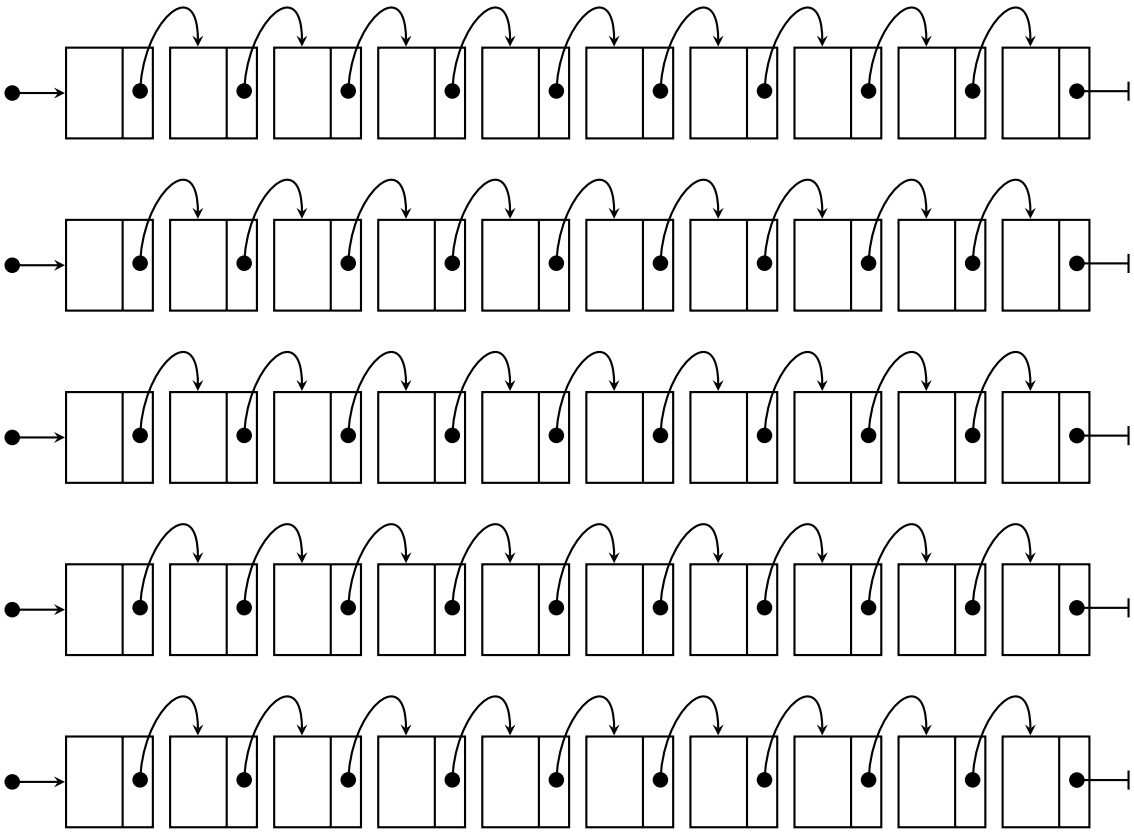
Aufgabe 4 BubbleSort *[Punkte: 4]*

Im Folgenden ist eine verkettete Liste mit Zahlenwerten abgebildet:



Die Liste soll mittels Bubblesort aufsteigend von vorne nach hinten sortiert werden.

(a) *(3 Punkte)* Tragen Sie im Folgenden ein, wie die Liste nach den einzelnen Durchläufen der äußeren Schleife von BubbleSort aussieht. Gehen Sie davon aus, dass der Algorithmus terminiert, sobald die Liste sortiert ist. Streichen Sie eventuell nicht benötigte Zeilen aus.



(b) *(1 Punkte)* Der BubbleSort-Algorithmus ließe sich auch so implementieren, dass die Liste von hinten nach vorne durchlaufen wird.

Was spricht bei der gezeigten Listenstruktur dagegen, den Algorithmus auf diese Weise zu implementieren? Begründen Sie in einem Satz.