Universität Stuttgart

Institut für Visualisierung und Interaktive Systeme

Übungsblatt 1

Datenstrukturen und Algorithmen (SS 2021)

Abgabe: Montag, 03.05.2021, 12:00 Uhr — Besprechung: ab Montag, 10.05.2021

Bitte lösen Sie die Übungsaufgabe in **Gruppen von 3 Bearbeiter*innen** und wählen Sie EINE Person aus der Gruppe aus, welche die Lösung in ILIAS als **PDF** als **Gruppenabgabe** (unter Angabe aller Gruppenmitglieder) einstellt. Bitte erstellen Sie dazu ein **Titelblatt**, das die Namen der Bearbeiter*innen, 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]*

Gegeben sind die folgenden Listen, damit Sie das *sequenzielle* und das *binären* Suchverfahren selbst anwenden können:

Gesuchtes Element: **11**

1	23	47	43	68	12	11	73	81	76	54	21	23	90	90
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Gesuchtes Element: **48**

12	13	23	47	48	68	69	73	81	83	84	85	87	90	94
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

(a) *(4 Punkte)* Nutzen Sie das in den Vorlesungsfolien/-aufzeichnungen vorgestellten Schema zur *sequenziellen* Suche und wenden Sie diese auf die zwei gegebenen Listen an. Führen Sie dabei die einzelnen Schritte auf, damit deutlich wird, ob die *sequenzielle* Suche möglich ist und wie diese schrittweise abläuft.

(b) *(3 Punkte)* Nutzen Sie das in den Vorlesungsfolien/-aufzeichnungen vorgestellten Schema zur *binären* Suche und wenden Sie diese ebenfalls auf die zwei gegebenen Listen an. Führen Sie dabei die einzelnen Schritte auf, damit deutlich wird ob die *binäre* Suche möglich ist und wie diese schrittweise abläuft.

[1] https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_fold_2387968.html

Gesuchtes Element: **11**

i)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	23	47	43	68	12	11	73	81	76	54	21	23	90	90

Gesuchtes Element: **48**

ii)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	13	23	47	48	68	69	73	81	83	84	85	87	90	94

a) Sequenzielle

i)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	23	47	43	68	12	11	73	81	76	54	21	23	90	90
1	**23**	47	43	68	12	11	73	81	76	54	21	23	90	90
1	23	**47**	43	68	12	11	73	81	76	54	21	23	90	90
1	23	47	**43**	68	12	11	73	81	76	54	21	23	90	90
1	23	47	43	**68**	12	11	73	81	76	54	21	23	90	90
1	23	47	43	68	**12**	11	73	81	76	54	21	23	90	90
1	23	47	43	68	12	**11**	73	81	76	54	21	23	90	90

ii)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	13	23	47	48	68	69	73	81	83	84	85	87	90	94
12	**13**	23	47	48	68	69	73	81	83	84	85	87	90	94
12	13	**23**	47	48	68	69	73	81	83	84	85	87	90	94
12	13	23	**47**	48	68	69	73	81	83	84	85	87	90	94
12	13	23	47	**48**	68	69	73	81	83	84	85	87	90	94

b) binäre

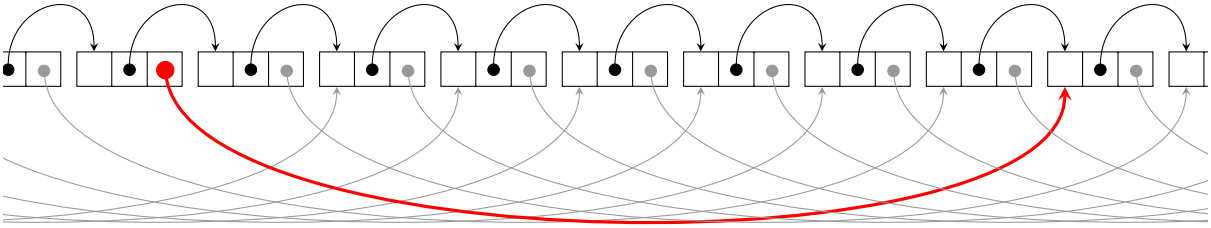
i) unmöglich, weil diese Liste nicht in der richtige Reihenfolge angeordnet.

ii)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
12	13	23	47	48	68	69	**73**	81	83	84	85	87	90	94
12	13	23	**47**	48	68	69	73	81	83	84	85	87	90	94
12	13	23	47	48	**68**	69	73	81	83	84	85	87	90	94
12	13	23	47	**48**	68	**69**	73	81	83	84	85	87	90	94

Aufgabe 2 Impl Verkettete Listen *[Punkte: 16]*

Im Folgenden ist eine verkettete Liste dargestellt. Zusätzlich zu der Verkettung mit dem direkten Nachbarn ist jeder Knoten auch noch mit – soweit vorhanden – seinem n-ten Nachbarn verbunden. Zur Verdeutlichung ist eine der Abkürzungskanten für den Fall $n = 8$ exemplarisch rot hervorgehoben:



Implementieren Sie die oben abgebildete Listenstruktur, bestehend aus einer Klasse `SpeedList<T>` (die eine Schnittstelle `ISpeedList` implementiert) und einer dazugehörigen Klasse `Node`, welche einen Knoten der Liste abbildet.

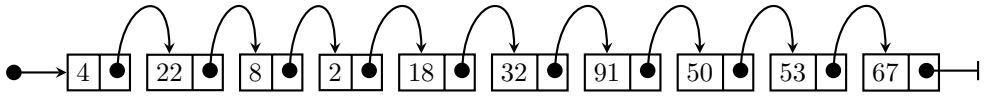
Hinweis: Die Verwendung von `java.util.LinkedList` etc. gibt in diesem Aufgabenblatt keine Punkte, da Sie die Implementierung selbst vornehmen sollen

In Ihrer Implementierung soll die Weite (n) der Abkürzungskanten an den Konstruktor übergegeben werden können. Wann immer möglich und sinnvoll, müssen die Abkürzungskanten verwendet werden, um die Navigation innerhalb der Liste zu beschleunigen.

Die Schnittstelle und die zu implementierende Klasse sind im Eclipse-Projekt zu dieser Aufgabe enthalten.

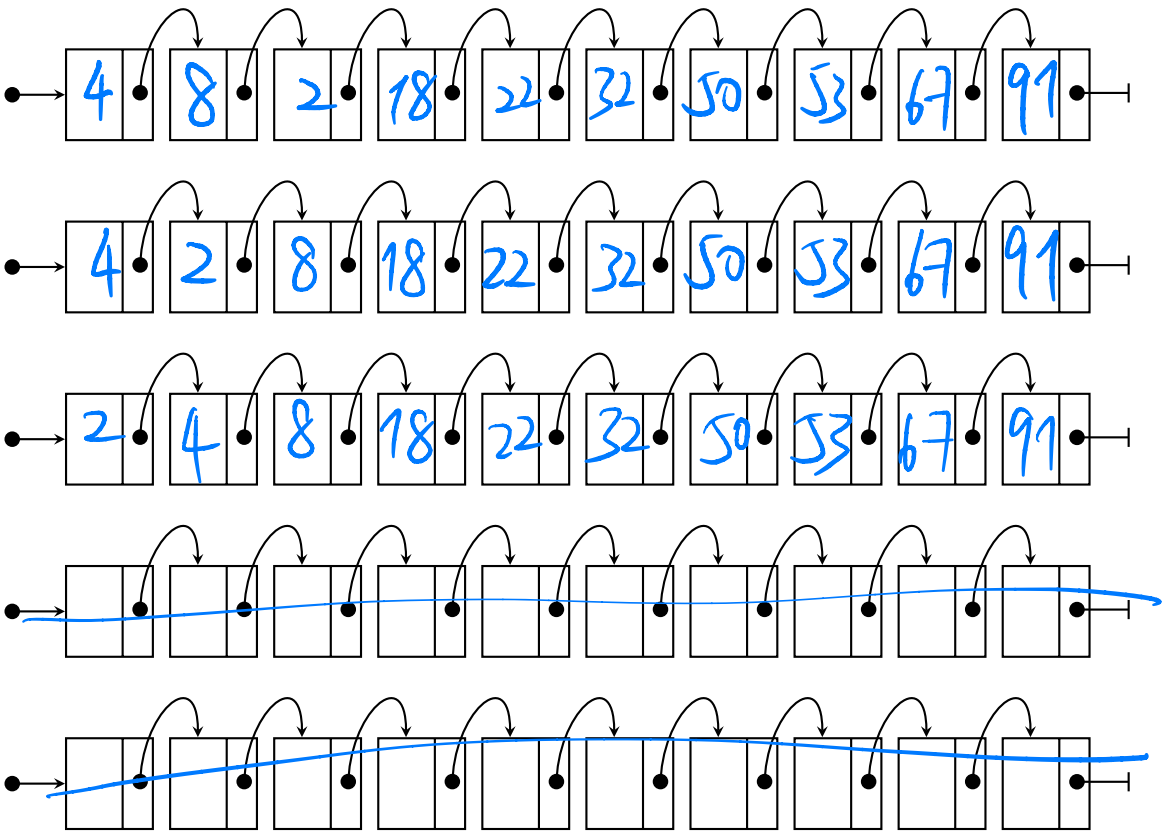
Aufgabe 3 BubbleSort *[Punkte: 4]*

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.

Aufgabe 4 Sortierverfahren Komplexität *[Punkte: 3]*

Im Folgenden sind die schnellsten Sortierverfahren für unterschiedliche Anwendungsfälle gesucht. Markieren Sie im Folgenden für jeden Fall, welches Sortierverfahren aus der Vorlesung die gegebenen Daten am schnellsten (aufsteigend) sortiert. Gibt es mehrere gleich schnelle Verfahren, dann markieren Sie alle. Begründen Sie außerdem ihre Auswahl und geben Sie dabei zu jedem Anwendungsfall an, welche Komplexität das ausgewählte Verfahren besitzt.

Fall 1: Gegeben sind absteigend sortierte Daten

☒ QuickSort ☐ SelectionSort ☐ BubbleSort ☒ MergeSort ☐ InsertionSort

Begründung/Komplexität: O(n log n)

Fall 2: Gegeben sind diese sortierten Daten => [5, 6, 8, 9, 11, 23, 32, 34, 58, 59, 123, 233, 436, 543]

☐ QuickSort ☐ SelectionSort ☒ BubbleSort ☐ MergeSort ☒ InsertionSort

Begründung/Komplexität: O(n)

Fall 3: Gegeben sind unsortierte („chaotische“) Daten

☒ QuickSort ☐ SelectionSort ☐ BubbleSort ☒ MergeSort ☐ InsertionSort

Begründung/Komplexität: O(n log n)