Check und Prepare

Vorbereitende Übungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 09

Version: 23. Januar 2022
Semesterübergreifend

V1 LinkedList

Für diese Aufgabe betrachten wir folgende Klasse für Listenelemente, die Sie auch schon in der Vorlesung kennengelernt haben.

```
1 public class ListItem<T> {
2
3   public T key;
4   public ListItem<T> next;
5 }
```

Alle nachfolgenden Aufgaben sollen dabei in folgender Klasse implementiert werden:

```
1 public class MyLinkedList<T> {
2
3   private ListItem<T> head;
4
5   public MyLinkedList() {
6     head = null;
7   }
8
9   // Insert your methods here
10 }
```

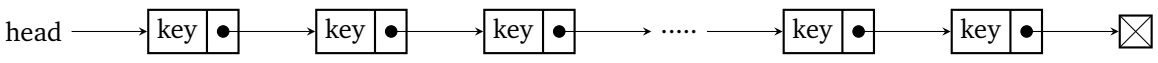

Abbildung 1: Eigene Linked List-Klasse auf Basis der Vorlesung

V1.1 Neues Element hinzufügen

Implementieren Sie die Methode `void add(T key)`. Diese bekommt einen neuen Schlüssel übergeben und erstellt ein neues Listenelement mit dem übergebenen Schlüssel, welches ganz am Ende der Liste angehängt wird.

Denken Sie an den Spezialfall, wenn noch kein Element in der Liste vorhanden ist.

V1.2 Element entfernen I

Implementieren Sie die Methode `void delete(int pos)`. Die Methode entfernt das Element an der Position `pos` aus

der Liste, wobei das erste Element die Position `0` besitzt. Ist `pos` keine gültige Position, so wirft die Methode eine `IndexOutOfBoundsException` mit der Botschaft `"Invalid position!"`.

V1.3 Element entfernen II

Implementieren Sie die Methode `void delete(T key)`. Die Methode entfernt das erste wertgleiche Vorkommen des Elements `key` aus der Liste. Sollte das Element nicht vorkommen, so bleibt die Liste unverändert.

V1.4 Drittletztes Element

Implementieren Sie die Methode `T beforeBeforeLast()`. Der Rückgabewert der Methode ist `null`, falls die Liste nicht mindestens drei Elemente hat. Ansonsten wird der Key vom drittletzten Element der Liste zurückgeliefert.

V1.5 Eins nach links bitte

Implementieren Sie die Methode `void ringShiftLeft()`. Die Methode verschiebt alle Listenelemente um eine Stelle nach links. Das heißt, dass das erste Element das neue letzte Element der Liste wird.

V1.6 Liste in Array

Implementieren Sie (ohne einfach die zugehörige Methode aus der Standardbibliothek aufzurufen) die Methode `T[] listIntoArray(Class<?> type)`. Die Methode wandelt die Liste in ein Array um, das heißt genau alle Schlüsselwerte der Liste sind in dem Array, welches zurückgegeben wird, in ursprünglicher Reihenfolge enthalten. Sind keine Schlüsselwerte in der Liste enthalten, so soll ein Array der Länge null zurückgegeben werden.

> **Tipp:** Ein Array des Typs `T` erstellen Sie am Besten auf folgende Weise:
>
> ```
> (T[])Array.newInstance(type, size)
> ```

V1.7 Liste in Listen

Implementieren Sie die Methode `MyLinkedList<MyLinkedList<T>> listInLists()`. Die Methode teilt die Liste in eine Liste von mehreren einelementigen Listen auf, wobei jeder Schlüsselwert der Eingabeliste, zu genau einem Schlüsselwert einer einelementigen Liste wird.

V1.8 Quadratzahlen aus der Liste entfernen

Implementieren Sie die Methode `void` deleteSquareNumbers(). Die Methode entfernt alle Elemente aus der Liste, deren Position in der Liste eine Quadratzahl ist, wobei das erste Listenelement Position `0` hat.[1]

V1.9 Listen in Liste

Implementieren Sie die Methode `MyLinkedList<T> listsInList(MyLinkedList<MyLinkedList<T>> lsts)`. Die Methode erstellt eine zusammenhängende Liste aus dem Parameter `lsts`. Dazu sollen alle Listen des Parameters `lsts` in der ursprünglichen Reihenfolge hintereinander angefügt werden und die daraus resultierenden neue Liste zurückgegeben werden. Vergleichen Sie dazu auch nochmal Aufgabe V1.7.

[1] 0 ist natürlich auch eine Quadratzahl

Übungsblatt 09: Vorbereitende Übungen von Nhan Huynh und Darya Nikitina

V2 Eigene verzeigerte Struktur in Racket

In Racket haben Sie Listen schon einige Male gesehen und benutzt. In dieser Aufgabe wollen wir uns nach dem Vorbild von Aufgabe V1 eine eigene verzeigerte Struktur erstellen. Dafür ist bereits folgende Struktur vorgegeben:

```
1 (define-struct lst-item (value  next))
```

Im Feld `value` des Structs wird der Schlüsselwert eines jeden `lst-items` gespeichert. Im Feld `next`, wird der Nachfolger eines `lst-items` gespeichert. Der Nachfolger ist entweder ebenfalls ein `lst-item` oder `null`, wenn es keinen Nachfolger gibt.

V2.1 Sortieren der Liste in aufsteigender Reihenfolge

Definieren Sie eine Funktion `sort-lst`.

Diese bekommt den Kopf einer Liste übergeben, sortiert die `values` der Elemente aufsteigend und liefert den Kopf dieser aufsteigend sortierten Liste zurück. Sie können davon ausgehen, dass die Liste nur Zahlen enthält. Benutzen Sie die Funktion `null?` um zu überprüfen, ob das letzte Element der Liste erreicht wurde. In diesem Fall gibt die Funktion `null?` dann `true` zurück, wenn damit das `next`-Feld, der `lst-item`-Struktur überprüft wird.

V3 Alternative LinkedList

In der Vorlesung haben Sie außerdem eine alternative `LinkedList` Implementierung kennengelernt. Diese zeichnet sich dadurch aus, dass anstelle eines Keys pro Item der Liste, ein Array von Keys mit einer vorher festgelegten Größe verwendet wird. Wir nennen diese Art von Listen in dieser Aufgabe `ArrayList`.

Gegeben sei dafür folgende Klasse für Listenelemente:

```
1 public class ArrayListItem<T> {
2
3   public T[] a;
4   public int n;
5   public ArrayListItem<T> next;
6 }
```

Alle nachfolgenden Aufgaben sollen dabei in folgender Klasse implementiert werden:

```
1 class ArrayList<T> {
2
3   private ArrayListItem<T> head;
4   private int N; // size of each array stored in the items
5
6   public ArrayList(int arraySize) {
7     head = null;
8     N = arraySize;
9   }
10
11  // Insert your methods here
12 }
```

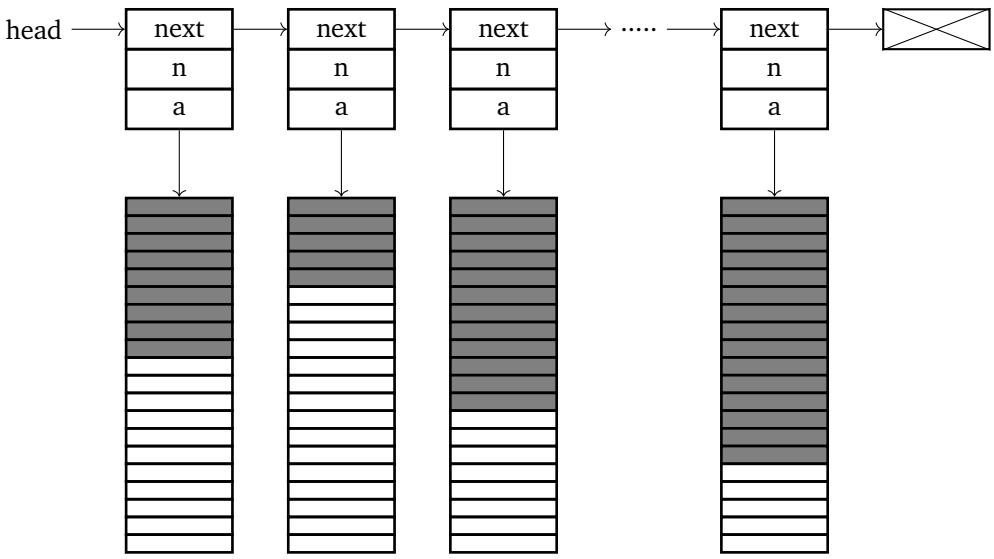

Abbildung 2: Eigene ArrayList-Klasse aus der Vorlesung

Hinweis: Mit Index i bezeichnen wir den Schlüssel, welcher sich an Index $i \mod n$ im Array des $\lfloor i/n \rfloor$-ten Listenelements befindet.

V3.1 contains-Methode ★☆☆

Implementieren Sie die Methode `int contains(T e)`. Diese durchsucht die Liste nachdem übergebenen Element `e` und gibt den ersten gefundenen Index in der Liste zurück, sofern es dort enthalten ist. Ist das übergebene Element nicht enthalten, soll stattdessen `-1` zurückgegeben werden.

V3.2 get-Methode ★☆☆

Implementieren Sie die Methode `T get(int index)`. Diese gibt das Element an dem übergebenen Index in der Liste zurück. Eine `IndexOutOfBoundsException` soll geworfen werden, wenn der übergebene Index kleiner null ist oder der Index die Größe der Liste überschreitet.

V3.3 set-Methode ★★☆

Implementieren Sie die Methode `void set(T e, int i)`. Diese bekommt ein Elemente vom Typ `T` und einen Index `i` übergeben und ersetzt das aktuelle Element am Index der Liste mit dem übergebenen. Eine `IndexOutOfBoundsException` soll geworfen werden, wenn der übergebene Index kleiner null ist oder der Index die Größe der Liste überschreitet.

V3.4 remove-Methode ★★☆

Implementieren Sie die Methode `void remove(int i)`. Diese bekommt einen Index `i` übergeben und entfernt das Element am übergebenen Index in der Liste. Alle nachfolgenden Elemente der Liste müssen daher um einen Index nach vorne verschoben werden, somit wird bei jedem nachfolgenden Element der Index dadurch um `1` geringer. War das zu entfernende Element das letzte Element in dem Array seines `ArrayListItems`, so muss der Verweis auf dieses `ArrayListItem` auf `null` gesetzt werden, da es nicht mehr verwendet wird. Eine `IndexOutOfBoundsException` soll geworfen werden, wenn der übergebene Index kleiner `0` ist oder der Index die Größe der Liste überschreitet.

V3.5 Komprimierung ★★★

Implementieren sie die Methode `void compress()`, welche die Liste komprimiert. Das heißt, nach Aufruf der Methode müssen alle internen Arrays bis auf das Letzte komplett befüllt sein. Das letzte Array muss hierbei mindestens ein Element ungleich `null` enthalten, das heißt es muss $n > 0$ gelten.

Abbildung 3: Beispiel für `compress()` mit $N = 3$