Check und Prepare

Lösungsvorschläge

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 }
```

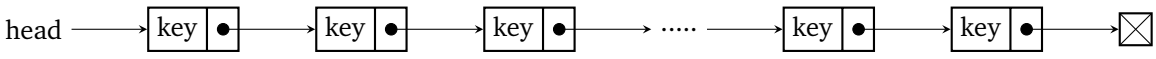

Abbildung 1: Eigene Linked List-Klasse auf Basis der Vorlesung

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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.

Lösungsvorschlag:

```java
/**
 * Adds the specified element to the end of this list.
 *
 * @param key the element to be added to this list
 */
public void add(T key) {
  ListItem<T> item = new ListItem<>();
  item.key = key;
  // List is empty, so the new list item object is the new head
  if (head == null) {
    head = item;
  } else {
    ListItem<T> p = head;
    // Last element is p and its successor is null
    while (p.next != null) {
      p = p.next;
    }
    p.next = item;
  }
}
```

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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!"`.

Lösungsvorschlag:

```java
/**
 * Removes the element at the specified position in this list.
 *
 * @param pos the index of the element
 * @throws IndexOutOfBoundsException if the index is out of range
 * (smaller than 0 or greater than the length of the list)
 */
public void delete(int pos) {
  // Out of bounds
  if (pos < 0) {
    throw new IndexOutOfBoundsException("Invalid position!");
  } else if (head == null) {
    return;
  }
  // If we delete the first element, we need to point the head of the list to its successor
  else if (pos == 0) {
    head = head.next;
    return;
  }
  // Search for the index of the element to delete
  int index = 0;
  for (ListItem<T> p = head; p.next != null; p = p.next, index++) {
    // Adjust the pointer of the predecessor to point the successor of this element
    if (index + 1 == pos) {
      p.next = p.next.next;
      return;
    }
  }
  // Index is greater than the length of the list
  throw new IndexOutOfBoundsException("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.

Lösungsvorschlag:

```java
/**
 * Removes the first occurrence of the specified element from this list, if there is one.
 *
 * @param key the element to be removed from this list, if there is one
 */
public void delete(T key) {
  // Cannot delete an element if the list is empty
  if (head == null) {
    return;
  }
  // If we delete the first element, the head of the list must point to its successor
  else if (head.key.equals(key)) {
    head = head.next;
  }
  // Search the element to delete
  for (ListItem<T> p = head; p.next != null; p = p.next) {
    // Adjust pointer of the predecessor to point to the successor of this element
    if (p.next.key.equals(key)) {
      p.next = p.next.next;
      // We are finished after we delete the element
      break;
    }
  }
}
```

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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.

Lösungsvorschlag:

```java
/**
 * Returns the key of the third last element of this list, if there is one.
 *
 * @return the key of the third last element of this list, if there is one
 */
public T beforeBeforeLast() {
  ListItem<T> p = head;
  // List must have at least 3 elements, else we return null
  if (p == null || p.next == null || p.next.next == null) {
    return null;
  }
  // Search for the third last element
  while (p.next.next.next != null) {
    p = p.next;
  }
  return p.key;
}
```

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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.

Lösungsvorschlag:

```
1  /**
2   * Shifts all element of this list on position to the left.
3   */
4  public void ringShiftLeft() {
5    // Cannot shift an empty list or a single element list
6    if (head == null || head.next == null) {
7      return;
8    }
9     /*
10     * The new pointer of the successor of the first element should be
11     * the successor of the last element.
12     * The new pointer of the successor of the last element should be
13     * pointing first element.
14     * The second element of the list is now the first element in the new list
15     */
16   ListItem<T> begin = head;
17   ListItem<T> next = head.next;
18   ListItem<T> end = head;
19   while (end.next != null) {
20     end = end.next;
21   }
22   begin.next = end.next;
23   end.next = begin;
24   head = next;
25 }
```

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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)
> ```

Lösungsvorschlag:

```java
/**
 * Converts a list to an array.
 *
 * @param type the class type of the new array
 * @return an array by converting a list
 * @see Array#newInstance(Class, int)
 */
@SuppressWarnings("unchecked")
public T[] listIntoArray(Class<?> type) {
  int size = 0;
  for (MyLinkedList<T> p = head; p != null; p = p.next) {
    size++;
  }
  T[] result = (T[]) Array.newInstance(type, size);
  int i = 0;
  for (ListItem<T> p = head; p != null; p = p.next, i++) {
    result[i] = p.key;
  }
  return result;
}
```

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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.

Lösungsvorschlag:

```java
/**
 * Returns a list of single element lists.
 *
 * @return a list of single element lists
 */
public MyLinkedList<MyLinkedList<T>> listInLists() {
  MyLinkedList<MyLinkedList<T>> result = new MyLinkedList<>();
  // Build the new list
  ListItem<MyLinkedList<T>> head = null;
  // References the last element of the list
  ListItem<MyLinkedList<T>> tail = head;
  for (ListItem<T> p = this.head; p != null; p = p.next) {
    ListItem<T> item = new ListItem<>();
    item.key = p.key;

    MyLinkedList<T> list = new MyLinkedList<>();
    list.head = item;

    ListItem<MyLinkedList<T>> itemList = new ListItem<>();
    itemList.key = list;

    // If head is empty, then the new created list item object is the new head
    if (head == null) {
      head = tail = itemList;
    }
    // Add elements to the last one
    else {
      tail.next = itemList;
      tail = tail.next;
    }
  }
  result.head = head;
  return result;
}
```

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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]

Lösungsvorschlag:

```
1  /**
2   * Removes all elements with square number indices of this list.
3   */
4  public void deleteSquareNumbers() {
5    // Index 0 and 1 are square numbers
6    if (head == null || head.next == null) {
7      return;
8    }
9    ListItem<T> p = head.next;
10   int position = 2;
11   int base = 2;
12   while (p.next != null) {
13     // Removing all other elements with square numbers
14     if (position == base * base) {
15       p.next = p.next.next;
16       base++;
17     } else {
18       p = p.next;
19     }
20     position++;
21   }
22   // Index 0 and 1 are square numbers
23   head = head.next.next;
24 }
```

[1] 0 ist natürlich auch eine Quadratzahl

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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.

Lösungsvorschlag:

```java
/**
 * Returns a single list by converting the specified list of lists.
 *
 * @param lsts the list of lists
 * @return a single list by converting the specified list of lists
 * @see MyLinkedList#add(Object)
 */
public MyLinkedList<T> listsInList(MyLinkedList<MyLinkedList<T>> lsts) {
  MyLinkedList<T> list = new MyLinkedList<>();
  // Build a new list
  ListItem<T> head = null;
  // References the last element of the list, faster performance
  ListItem<T> tail = head;
  // Iterate over all MyLinkedList
  for (ListItem<MyLinkedList<T>> l = lsts.head; l != null; l = l.next) {
    // Iterate over ListItem
    for (ListItem<T> p = l.key.head; p != null; p = p.next) {
      ListItem<T> item = new ListItem<>();
      item.key = p.key;
      // If head is empty, then the new created list item object is the new head
      if (head == null) {
        head = tail = item;
      }
      // Add elements to the last one
      else {
        tail.next = item;
        tail = tail.next;
      }
    }
  }
  list.head = head;
  return list;
}
```

Übungsblatt 09: Lösungsvorschläge 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.

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```
(define-struct lst-element (value next))

;; Type: number (list of lst-element) -> (list of lst-element)
;; Returns: a list of lst-element with a new element at the right position
;; inserted in it (so that this element is sorted)
(define (insert value lst)
  (if (or (null? lst) (< value (lst-element-value lst)))
      (make-lst-element value lst)
      (make-lst-element (lst-element-value lst)
                        (insert value (lst-element-next lst)))))

;; Tests
(check-expect (insert 2 (make-lst-element 1 (make-lst-element 3 null)))
  (make-lst-element 1 (make-lst-element 2 (make-lst-element 3 null))))
(check-expect (insert 1 (make-lst-element 2 (make-lst-element 3 null)))
  (make-lst-element 1 (make-lst-element 2 (make-lst-element 3 null))))
(check-expect (insert 1 (make-lst-element 2 (make-lst-element 3 null)))
  (make-lst-element 1 (make-lst-element 2 (make-lst-element 3 null))))

;; Type: (list of lst-element) -> (list of lst-element)
;; Returns: a sorted list of lst-element (via insertion sort)
(define (sort-lst lst)
  (if (null? lst) lst
      (insert (lst-element-value lst)
              (sort-lst (lst-element-next lst)))))

;; Tests
(check-expect (sort-lst (make-lst-element 1 (make-lst-element 2
  (make-lst-element 3 null))))
    (make-lst-element 1 (make-lst-element 2 (make-lst-element 3 null))))
(check-expect (sort-lst (make-lst-element 2 (make-lst-element 1
  (make-lst-element 3 null))))
    (make-lst-element 1 (make-lst-element 2 (make-lst-element 3 null))))
(check-expect (sort-lst (make-lst-element 3 (make-lst-element 2
  (make-lst-element 1 null))))
    (make-lst-element 1 (make-lst-element 2 (make-lst-element 3 null))))
```

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 }
```

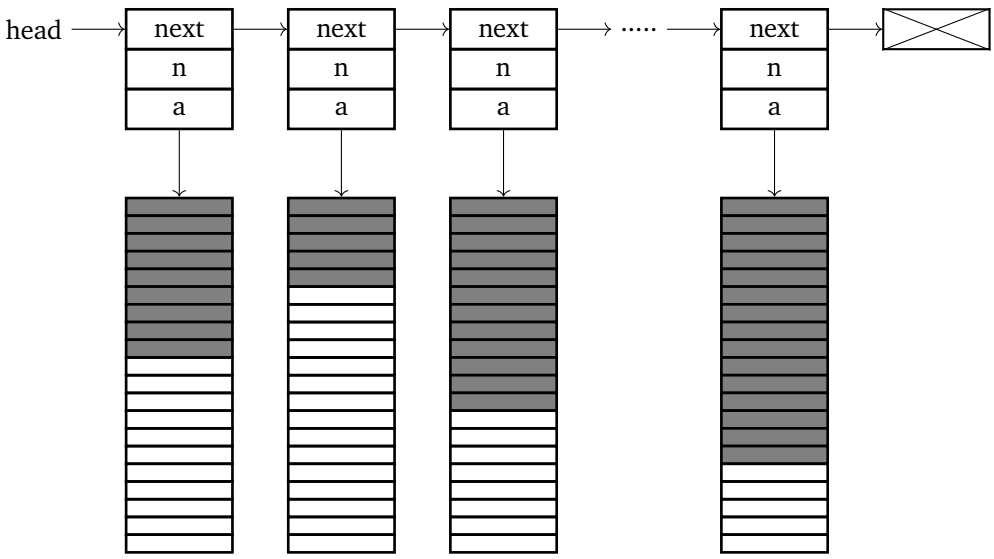

Abbildung 2: Eigene ArrayList-Klasse aus der Vorlesung

Übungsblatt 09: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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.

Lösungsvorschlag:

```
1  /**
2   * Returns the index of the element, if this array list contains it.
3   *
4   * @param element the element we want to find in the array list
5   * @return index the index of the first equal element to the given element
6   * or -1, if it the element is not in the array list
7   */
8  int contains(T element) {
9    int index = 0;
10   // Find correct array
11   for (ArrayListItem<T> p = head; p != null; p = p.next) {
12     // Check elements of the array
13     for (int i = 0; i < p.n; i++) {
14       if (p.a[i].equals(element)) {
15         index += i;
16         return index;
17       }
18     }
19     // Go to the next array
20     index += p.n;
21   }
22   return -1;
23 }
```

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.

Lösungsvorschlag:

```
1  /**
2   * Returns the element at the given index.
3   *
4   * @param index the index of the element
5   * @return the element at the given index
6   * @throws IndexOutOfBoundsException if the index is out of range (smaller than 0 or
        greater than
7   * the array)
8   */
9  T get(int index) {
10   ArrayListItem<T> p = head;
11   // Index starts at 0, but p.n by 1, if there are elements in the array
12   int pos = index + 1;
13   // Find the correct array
14   while (p != null && pos > p.n) {
15     pos -= p.n;
16     p = p.next;
17   }
18   pos--;
19   // Check for valid index
20   if (p == null || pos < 0) {
21     throw new IndexOutOfBoundsException(index);
22   }
23   return p.a[pos];
24 }
```

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.

Lösungsvorschlag:

```
1  /**
2   * Replaces the element at the given index with the specified element.
3   *
4   * @param e element to be stored at the specified position
5   * @param i index of the element to replace
6   * @throws IndexOutOfBoundsException if the index is out of range (smaller than 0 or
        greater than
7   * the array)
8   */
9  void set(T e,int i){
10     ArrayListItem<T> p=head;
11     // Index starts at 0, but p.n by 1, if there are elements in the array
12     int pos=i+1;
13     // Find the correct array
14     while(p!=null&&pos>p.n){
15     pos-=p.n;
16     p=p.next;
17     }
18     pos--;
19     // Check for valid index
20     if(p==null||pos< 0||p.a[pos]==null){
21     throw new IndexOutOfBoundsException(i);
22     }
23     p.a[pos]=e;
24 }
```

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.

Lösungsvorschlag:

```java
/**
 * Removes the element at the given index.
 *
 * @param i the index of the element to be removed
 * @throws IndexOutOfBoundsException if the index is out of range (smaller than 0 or
     greater than
 * the array)
 */
void remove(int i) {
  ArrayListItem<T> p = head;
  // Predecessor
  ArrayListItem<T> px = null;
  // Index start by 0, but p.n by 1, if there are elements in the array
  int pos = i + 1;
  // Find correct array
  while (p != null && pos > p.n) {
    pos -= p.n;
    px = p;
    p = p.next;
  }
  pos--;
  // Check for valid index
  if (p == null || pos < 0) {
    throw new IndexOutOfBoundsException(i);
  }
  // Shift elements to the left
  p.n--;
  for (int j = pos; j < p.n; j++) {
    p.a[j] = p.a[j + 1];
  }
  p.a[p.n] = null;
  // If the array is empty
  if (p.n == 0) {
    if (px == null) {
      head = head.next;
    } else {
      px.next = null;
    }
  }
}
```

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 $N = 3$

Lösungsvorschlag:

```java
/**
 * Compresses the array.
 */
void compress() {
  ArrayListItem<T> compressed = head;
  int i = 0;
  for (ArrayListItem<T> p = head; p != null; p = p.next) {
    int j = 0;
    for (T element : p.a) {
      // Skip element, if it is null
      if (element == null) {
        j++;
        continue;
      }
      // If the array is full, reset
      if (i == compressed.a.length) {
        compressed.n = i;
        compressed = compressed.next;
        i = 0;
      }
      // Compress
      final T e = p.a[j];
      p.a[j] = null;
      compressed.a[i] = e;
      i++;
      j++;
    }
  }
  // Adjust the last array list item
  compressed.n = i;
  compressed.next = null;
}
```