Funktionale und objektorientierte Programmierkonzepte

TECHNISCHE UNIVERSITÄT DARMSTADT

Präsenz-Sprechstunde B

Simon Hock, Nhan Huynh, Daniel Mangold

Überblick

TECHNISCHE UNIVERSITÄT DARMSTADT

Rekursion [25min]
    Vorgehensweise bei rekursiven Aufgaben

Arbeitsphase [70min]

ListItem [5min]

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class ListItem<T> {
    public T key;
    public ListItem<T> next;

    public ListItem() {}
}
```

```java
public class MyLinkedList<T> {

    public ListItem<T> head;
}
```

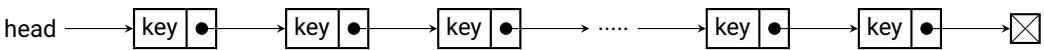


Abbildung: Eigene Linked List-Klasse auf Basis der Vorlesung

Analogie - Perlenkette

TECHNISCHE UNIVERSITÄT DARMSTADT

- `MyLinkedList`: Perlenkette
- `head`: Anfang der Perlenkette
- `key`: Perle
- `next`: Verbindung zwischen Perlen

Abbildung: Perlenkette

Rekursion

TECHNISCHE UNIVERSITÄT DARMSTADT

„Eine Rekursion ist eine Funktion, die sich selbst aufruft. Dabei wird versucht das Problem auf eine einfachere Variante des **gleichen** Problems rekursiv zu reduzieren, welches dann gelöst wird.“

Vorgehensweise bei rekursiven Aufgaben

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Identifiziere, wie das Problem auf eine einfachere Variante des Problems zerlegt werden kann und direkt gelöst werden kann
   - Teillösungen der Gesamtlösung
2. Das kleinste Problem ist ein Rekursionsanker
3. Überlege, wie die kleinen Probleme kombiniert werden können, um die ursprüngliche Problemstellung zu lösen
4. Kombiniere die Teillösungen zu einer Gesamtlösung

- Wir möchten zwei Listen miteinander verknüpfen.
- Die Verknüpfung ist folgendermaßen definiert:
  1. Sei d_i ein Element aus der Zielliste, s_i ein Element aus der Quellliste.
  2. Falls das `BiPredicate` für (d_i, s_i) `true` zurückliefert, so wird das Element vor d_i in der Zielliste eingefügt.
     2.1 Wiederhole 2. mit d_{i+1} und starte bei dem zuletzt betrachteten s_i.
  3. Ansonsten wird das nächste Element von der Zielliste betrachtet s_{i+1} und wiederhole 2.
  4. Wir sind fertig, sobald alle Elemente der Quellliste betrachtet wurden.
  5. Falls wir keine Elemente aus der Zielliste mehr betrachten, aber noch Elemente aus der Quelliste betrachtet werden müssen, dann werden diese am Ende der Zielliste angehängt.

Parameter

TECHNISCHE UNIVERSITÄT DARMSTADT

- `MyLinkedList<U> otherList`: Quelliste
- `BiPredicate<? super T, ? super U> biPred`: Prädikat bestimmt, wo das Element aus der Quelliste in die Zielliste eingefügt werden soll
- `Function<? super U, ? extends T> fct`: Wandelt den Typ eines Elements aus der Quellliste in den Typ eines Elements der Zielliste um
- `Predicate<? super U> predU`: Prädikat bestimmt, ob ein Element aus der Quelliste überhaupt aufgenommen werden darf. Ansonsten wird eine `MyLinkedListException` geworfen

```
1 public <U> void mixinRecursively(MyLinkedList<U> otherList,
2  BiPredicate<? super T, ? super U> biPred, Function<? super U, ? extends T>
3  fct, Predicate<? super U> predU) throws MyLinkedListException { ... }
```

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Falls die Summe der beiden Elemente im `biPred` ungerade ist, so wird das Element zur Zielliste aufgenommen
- Zuerst müssen wir die Position von der 1 bestimmen

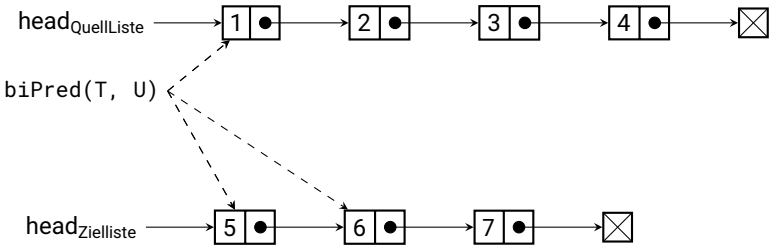
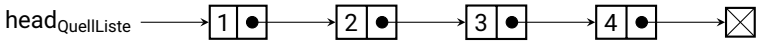


Abbildung: Beispiel H2.2 - Bestimme Position von 1

- Füge die 1 hinzu.




```
biPred(T, U)
```

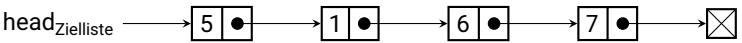


Abbildung: Beispiel H2.2 - Füge 1 hinzu

- Bestimme Position von 2.

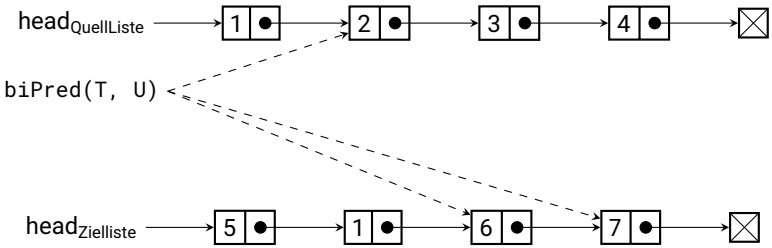


Abbildung: Beispiel H2.2

- Füge 2 hinzu.

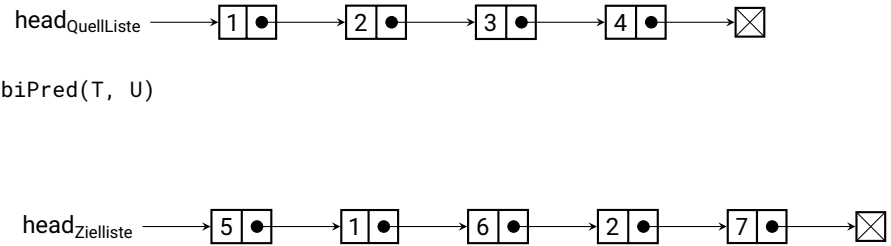


Abbildung: Beispiel H2.2 - Füge 2 hinzu

- Bestimme Position von 3 hinzu.

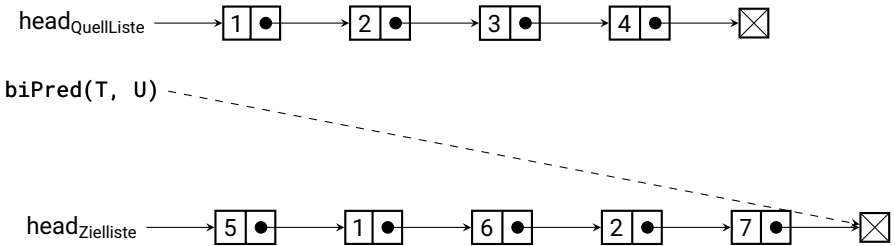


Abbildung: Beispiel H2.2 - Füge 2 hinzu

- Wir sind am Ende der Zielliste $\rightarrow$ Füge die restlichen Elemente von der Quellliste in die Zielliste hinzu.
- Wir sind nun fertig.

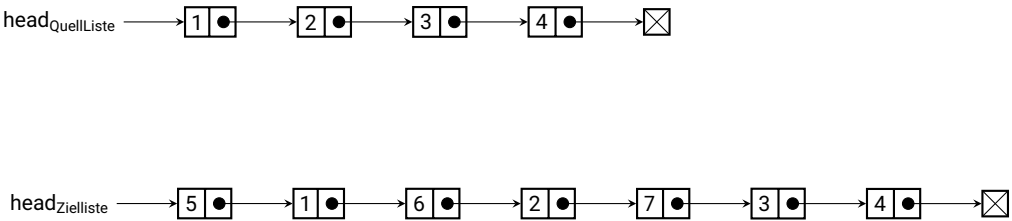


Abbildung: Beispiel H2.2 - Füge die restlichen Elemente hinzu

Welche Fälle müssen wir generell bei ListItem betrachten?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Operation wird in der Mitte ausgeführt
- Operation wird am Anfang ausgeführt (Sonderfall einmalig)
- Operation wird am Ende ausgeführt (Sonderfall einmalig)

Operation wird in der Mitte ausgeführt

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Identifiziere, wie das Problem in eine einfachere Variante des Problems zerlegt und direkt gelöst werden kann
- Wir betrachten ein Element
- Falls das Prädikat zutrifft, dann erstelle eine Liste und füge das Element hinzu
- Ansonsten mache nichts

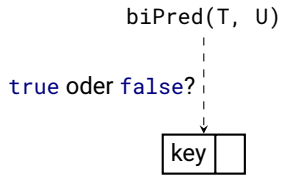


Abbildung: Identifiziere Problem

2. Das kleinste Problem ist ein Rekursionsanker

- Kleinstes Problem: Quellliste ist leer $\rightarrow$ Wir müssen nichts mehr machen

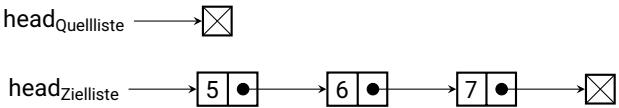


Abbildung: Kleinstes Problem

```java
private <U> void mixinRecursivelyHelper(MyLinkedList<U> otherList,
                    BiPredicate<? super T, ? super U> biPred,
                    Function<? super U, ? extends T> fct,
                    Predicate<? super U> predU,
                    ListItem<U> pSrc,
                    ListItem<T> pDest,
                    int index) throws MyLinkedListException {
  if (pSrc == null) {
    return;
  }
    ...
}
```

3. Überlege, wie die kleinen Probleme kombiniert werden können, um die ursprüngliche Problemstellung zu lösen

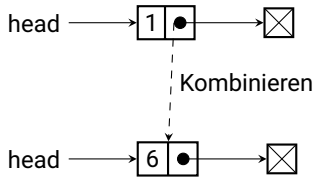


Abbildung: Visualisierung von Kombinieren

4. Kombiniere die Teillösungen zu einer Gesamtlösung

- Wir können nur Elemente von der Quellliste einfügen, sofern das Prädikat `predU` erfüllt wird

```
1 // Check if source element should be inserted
2 if (!predU.test(key)) {
3   throw new MyLinkedListException(index, key);
4 }
```

4. Kombiniere die Teillösungen zu einer Gesamtlösung
   - Am besten arbeiten wir mit dem Nachfolger, damit wir immer vorher einfügen können
   - Einfügen erfolgt nur, wenn `biPred` erfüllt ist

```
1  else if (pDest.next != null) {
2    T element = pDest.next.key;
3    if (biPred.test(element, key)) {
4      T mapped = fct.apply(key); ListItem<T> item = new ListItem<>(mapped);
5      // Element will be inserted before the element that fulfills the
6      // predicate
7      // Before -> New Element -> Predicate true element
8      item.next = pDest.next;
9      pDest.next = item;
10     // Update pointer since we added an element from the other list to this
11     // list
12     pSrc = pSrc.next; pDest = pDest.next;
13   }
```

- Zum Schluss müssen wir zum nächsten Element gehen und wiederholen das ganze erneut

```
1 // Iterate over the next elements
2 mixinRecursivelyHelper(otherList, biPred, fct, predU,
3   pSrc, pDest.next,
4   index + 1);
```

Was fehlt noch?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Spezialfälle mit erstem und letztem Element
- Wenn wir am Ende der Zielliste angekommen sind, dann fügen wir alle Elemente danach ein
- Der `else`-Teil wird nur ausgeführt, wenn `pDest.next == null` gilt, also wenn wir am Ende der Zielliste angekommen sind

```
1 else {
2   // Case current == null only occurs if we reached the tail of the list
3   T mapped = fct.apply(key);
4   pDest.next = new ListItem<>(mapped);
5   pSrc = pSrc.next;
6 }
```

```java
private <U> void mixinRecursivelyHelper(MyLinkedList<U> otherList,
                    BiPredicate<? super T, ? super U> biPred,
                    Function<? super U, ? extends T> fct,
                    Predicate<? super U> predU,
                    ListItem<U> pSrc,
                    ListItem<T> pDest,
                    int index) throws MyLinkedListException {
  // We have to insert all elements from the other list until there are no
     elements left
  if (pSrc == null) {
    return;
  }
  U key = pSrc.key;
  // Check if source element should be inserted
```

```
14    if (!predU.test(key)) {
15      throw new MyLinkedListException(index, key);
16    } else if (pDest.next != null) {
17      T element = pDest.next.key;
18      if (biPred.test(element, key)) {
19        T mapped = fct.apply(key); ListItem<T> item = new ListItem<>(mapped);
20        // Element will be inserted before the element that fulfills the
21        // predicate
22        // Before -> New Element -> Predicate true element
23        item.next = pDest.next;
24        pDest.next = item;
25
26        // Update pointer since we added an element from the other list to this
27        // list
28        pSrc = pSrc.next; pDest = pDest.next;
29      }
```

```
30    } else {
31      // Case current == null only occurs if we reached the tail of the list
32      T mapped = fct.apply(key);
33      pDest.next = new ListItem<>(mapped);
34      pSrc = pSrc.next;
35    }
36    // Iterate over the next elements
37    mixinRecursivelyHelper(otherList, biPred, fct, predU,
38      pSrc, pDest.next,
39      index + 1);
40  }
```

Spezialfall: Erstes Element

TECHNISCHE UNIVERSITÄT DARMSTADT

- Momentan funktioniert die Methode nur für alle Elemente nach dem ersten Element...
- Wie können wir das ändern? $\rightarrow$ Wir fügen ein „Dummy“-Element vor der eigentlichen Liste ein, damit das erste Element tatsächlich das zweite ist und somit unsere Implementierung funktioniert!
- Der Nachfolger vom „Dummy“-Element enthält das ggf. „neue“ erste Element der Liste

```java
public <U> void mixinRecursively(MyLinkedList<U> otherList,
                BiPredicate<? super T, ? super U> biPred,
                Function<? super U, ? extends T> fct,
                Predicate<? super U> predU) throws MyLinkedListException {
  // Cannot merge other list if its empty
  if (otherList.head == null) return;
  // Extend the list item by one dummy node to simplify the check with the
  // successor node
  ListItem<T> dummy = new ListItem<>(); dummy.next = head;
  mixinRecursivelyHelper(otherList, biPred, fct, predU, otherList.head, dummy
      , 0);
  head = dummy.next; // Restore head
}
```

Selbstständiges Arbeiten