Übung Algorithmen und Datenstrukturen

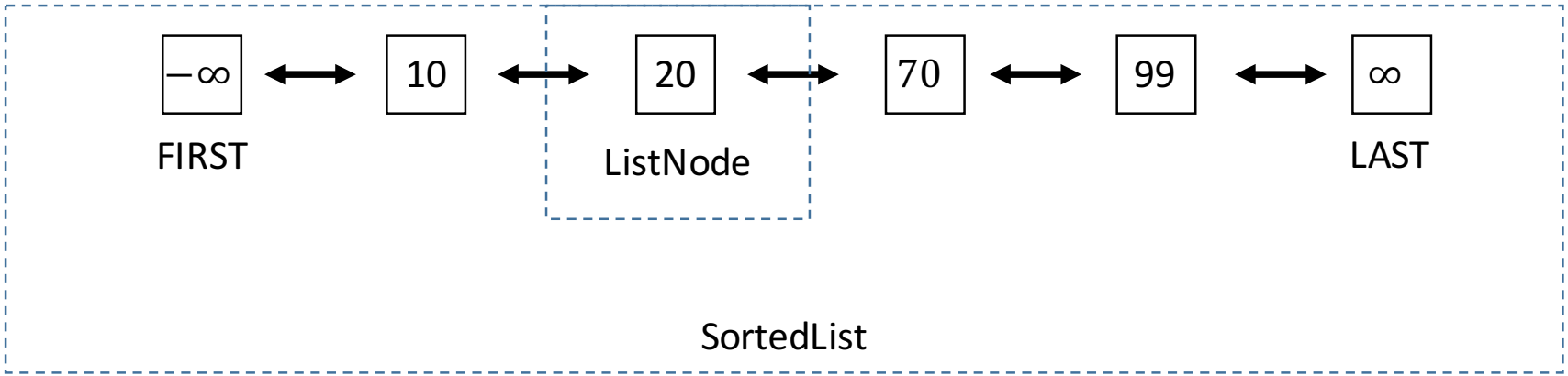
Sommersemester 2016

Patrick Schäfer, Humboldt-Universität zu Berlin

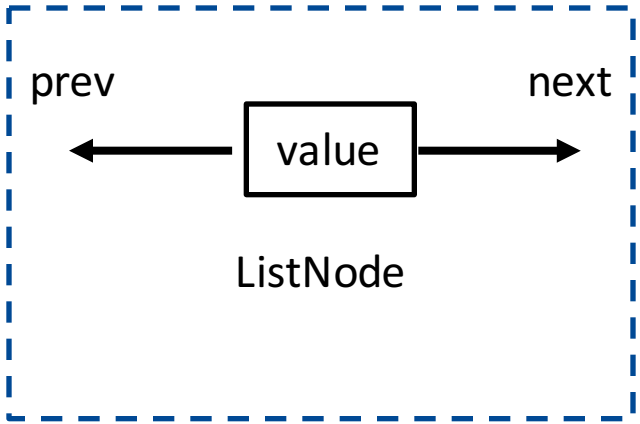
Agenda

1. Fragen zum zweiten Übungsblatt?
2. Vorbesprechung des dritten Übungsblatts
3. Beispielaufgaben

Doppelt verkettete sortierte Liste



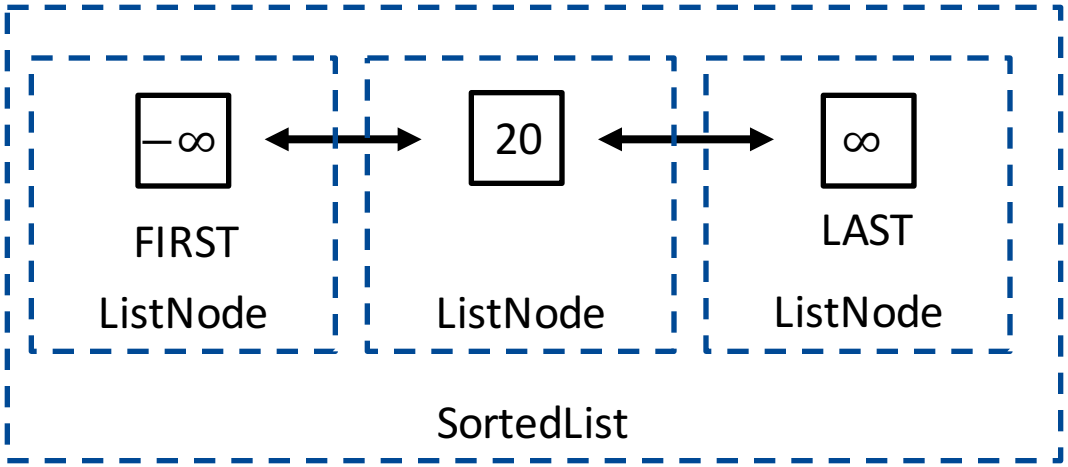
Knoten: ListNode




```java
public class ListNode {
  // Der Wert des Knoten
  public int value;

  // Zeiger auf den Nachfolger
  // und den Vorgänger
  public ListNode next;
  public ListNode prev;
}
```

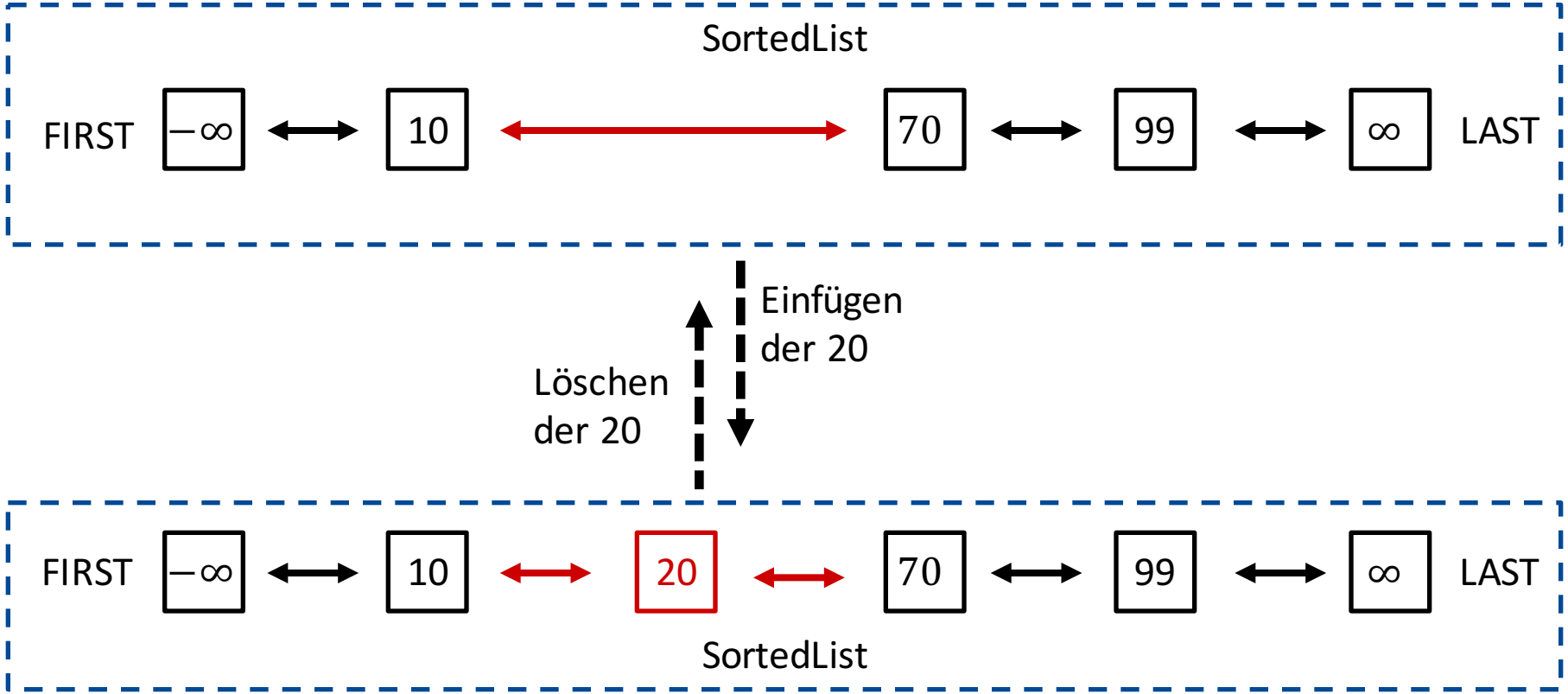
Doppelt verkettete sortierte Liste: SortedList



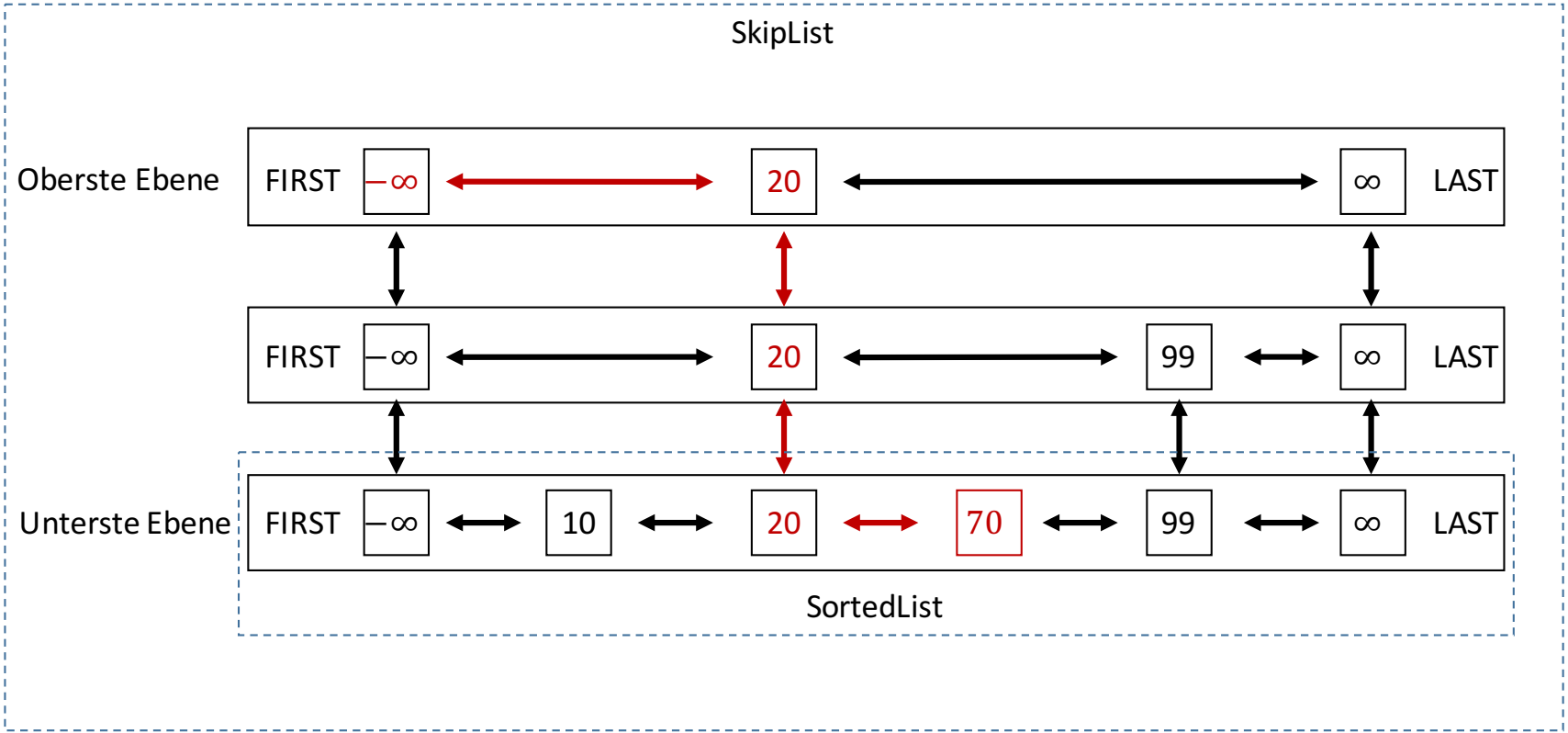
- Die leere Liste enthält immer den Start- und Endknoten.

```
public class SortedList
        implements LinkedList {
  public ListNode first;
  public ListNode last;
}
```

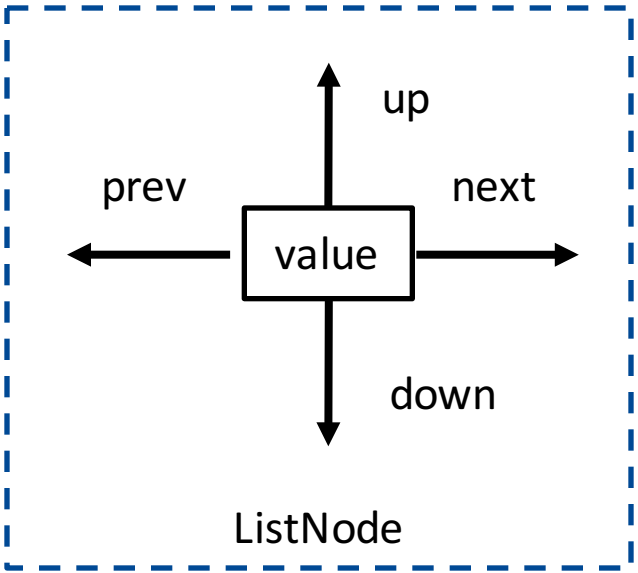
Einfügen


SortedList
FIRST
−∞
10
70
99
∞
LAST
Einfügen
der 20
Löschen
der 20
FIRST
−∞
10
20
70
99
∞
LAST
SortedList

Skip-Listen


SkipList
Oberste Ebene
FIRST
−∞
20
∞
LAST
FIRST
−∞
20
99
∞
LAST
Unterste Ebene
FIRST
−∞
10
20
70
99
∞
LAST
SortedList

Knoten: ListNode

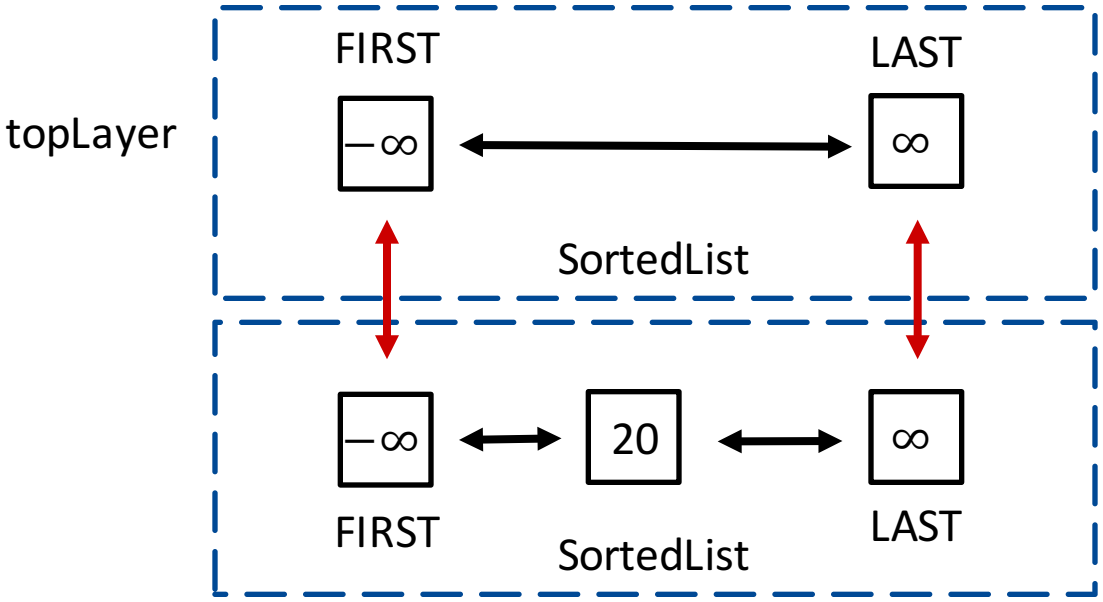



```java
public class ListNode {
  // Der Wert des Knoten
  public int value;

  // Zeiger auf den Nachfolger
  // und den Vorgänger
  public ListNode next;
  public ListNode prev;

  // Zeiger auf die obere
  // und untere Ebene
  public ListNode up;
  public ListNode down;
}
```

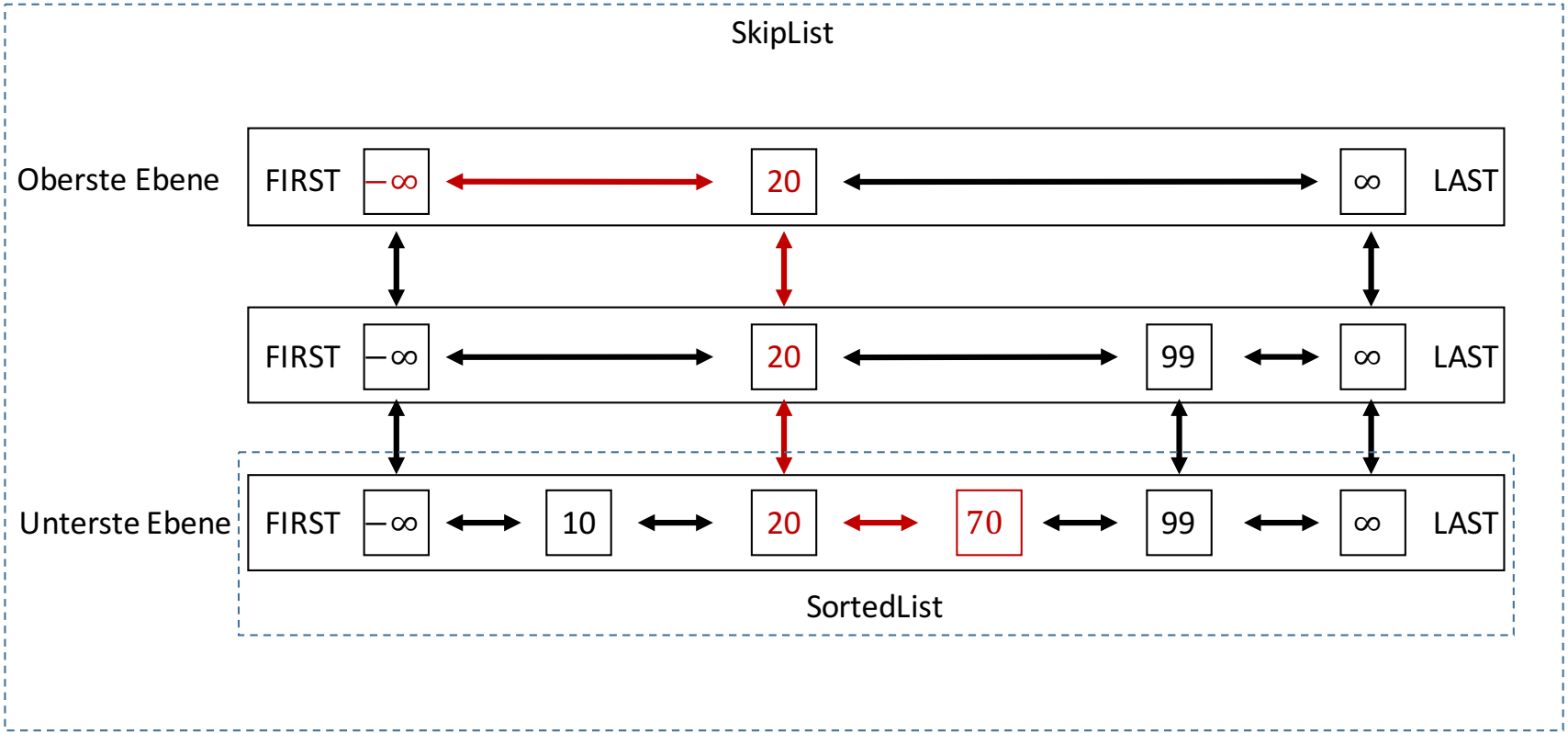
Skip-Liste: SkipList



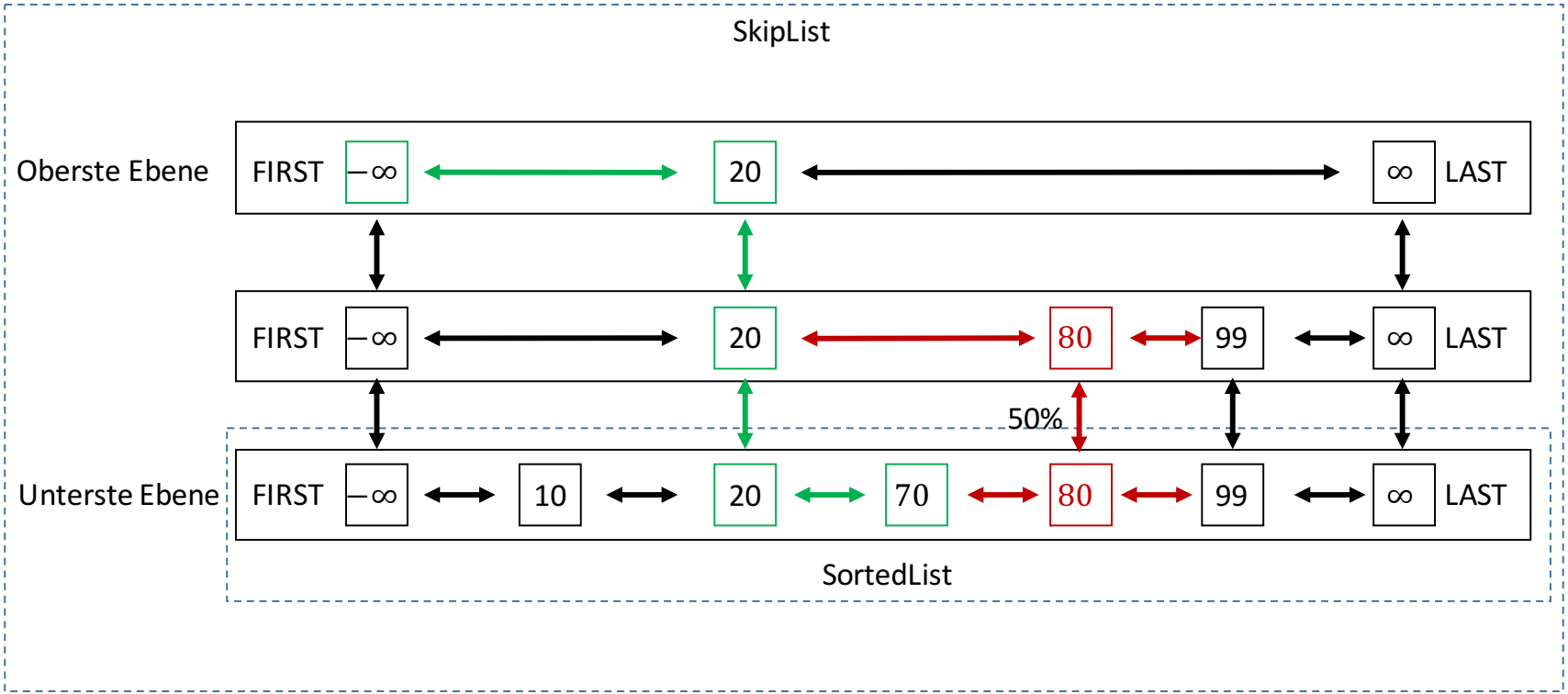

```
public class SkipList
        implements LinkedList {
  SortedList topLayer;
}
```

- Jede Ebene repräsentiert eine sortierte doppelt verkettete Liste.
- Start- und Endknoten benachbarter Ebenen sind miteinander verbunden.
- Kommt ein Knoten auf einer oberen Ebene vor, so kommt er auch auf allen darunter liegenden Ebenen vor.
- Alle Werte kommen auf der untersten Ebene vor.

Skip-Listen: Suchen

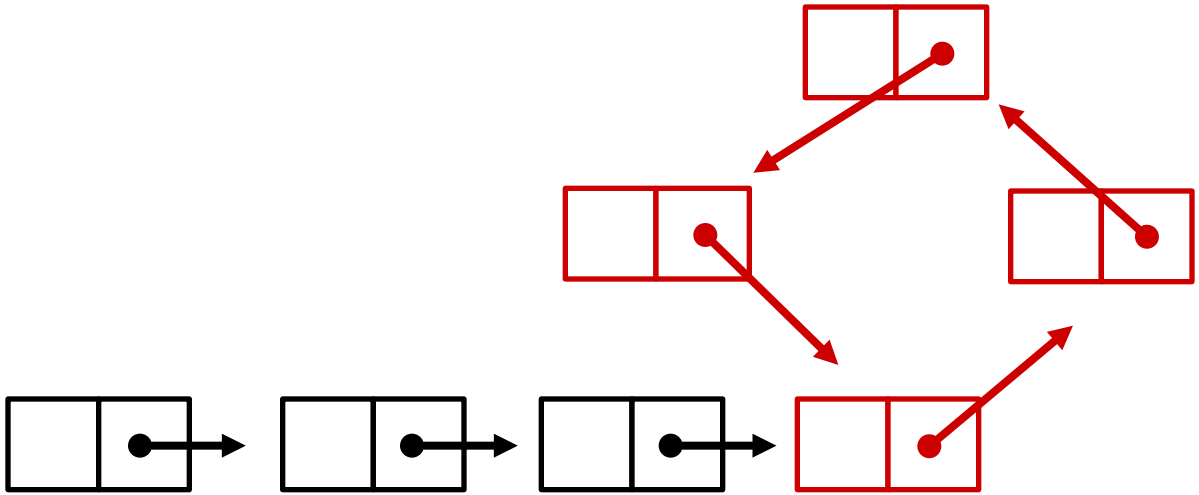


Skip-Listen: Einfügen



Übungsaufgabe: Kreise suchen

Gegeben sei eine einfach verkettete Liste. Nun möchte man seine Liste dahingehend überprüfen, ob es (z.B. durch einen Bug in der Implementierung) zu Schleifen gekommen ist. Wie geht das möglichst effizient?



Übungsaufgabe: Kreise suchen

Algorithmus $findeKreis(l)$

Input: Liste l
Output: Boolean

```
(1)   slow = l.first()
(2)   fast= l.first()
(3)   while (fast != NULL and fast.next != NULL) do
(4)     slow = slow.next       // ein Schritt
(5)     fast = fast.next.next  // zwei Schritte
(6)     if (slow == fast) then
(7)       return true
(8)     end if
(9)   end while
(10)  return false;
```

- Zwei Pointer durchlaufen die List. In jeder Runde geht ein der einen Schritt und der zweite zwei Schritte.
- Sobald der schnelle Pointer den langsamen Pointer überrundet gibt es eine Schleife.
- Falls der schnelle Pointer bei NULL ankommen ist, gibt es keine Schleife.

Tiefensuche

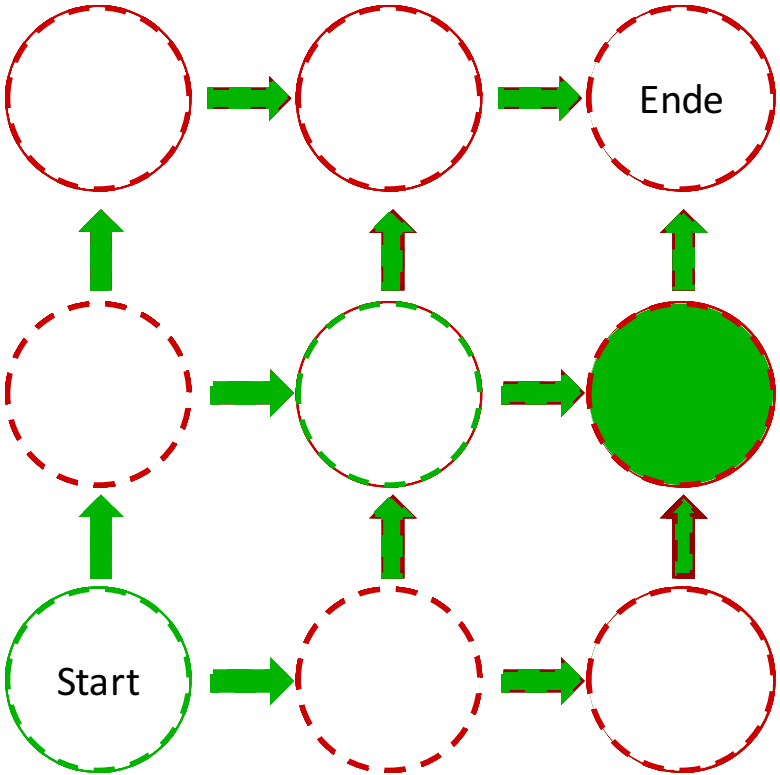
Algorithmus $dfs(start, besucht)$

Input: Array besucht, Integer start
Output:

```
(1)  besucht[start] = true;          // start wurde besucht
(2)  foreach (knoten in nachbarn(start)) {  // alle Nachbarn
(3)    if (!besucht[knoten]) then // neuer Knoten
(4)      DFS(knoten, besucht);
(5)    end if
(6)  end foreach
```

- Verfolgt einen Pfad so weit wie möglich.
- Die zu verarbeitenden Knoten werden in LIFO-Reihenfolge (last-in first-out / Stack) verarbeitet.
- Es gibt zwei Verarbeitungsstufen:
  1. Wenn der Knoten als Nachfolger entdeckt wird.
  2. Wenn der Knoten abgearbeitet wird.

Beispiel: Tiefensuche



Algorithmus $dfs(start, besucht)$

Input: Array besucht, Integer start
Output:

```
(1)  besucht[start] = true;          // start wurde besucht
(2)  foreach (knoten in nachbarn(start)) {  // alle Nachbarn
(3)    if (!besucht[knoten]) then // neuer Knoten
(4)      DFS(knoten, besucht);
(5)    end if
(6)  end foreach
```

Aufgabe: Dynamische Programmierung

- Wie kann geprüft werden, ob das Ziel erreicht wurde?
- Wie wird der gefundene Weg zurückgegeben?
- Wie könnten Hindernisse beachtet werden?

Übungsaufgabe: Suche

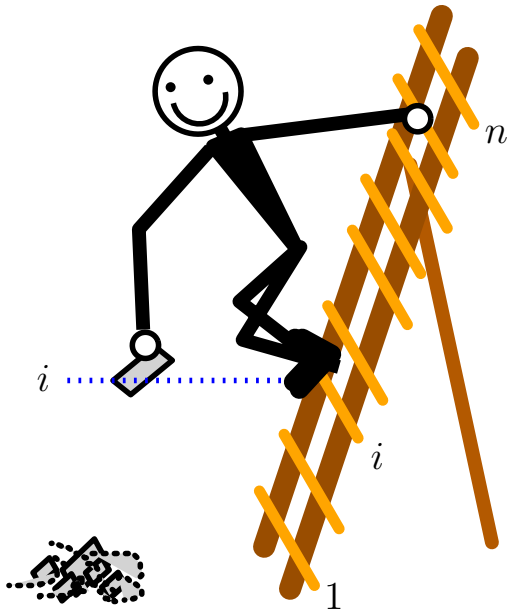
Als Redakteur einer Handy-Zeitschrift möchten Sie ein neues Smartphone auf seine Bruchtauglichkeit testen. Sie haben k Smartphones und eine Leiter mit n Sprossen, durchnummeriert von 1 bis n. In jedem Versuch steigen Sie auf eine Sprosse und lassen ein Smartphone gerade nach unten fallen. Entweder das Smartphone übersteht den Fall völlig unbeschadet oder es bricht auseinander.

Sie möchten die höchste Sprosse ermitteln, bei der ein Smartphone nicht bricht.

Da der Redaktionsschluss naht, stehen Sie unter Zeitdruck und möchten die Anzahl der Testversuche minimieren. Anderseits kosten Smartphones viel Geld und ihre Anzahl muss auch minimiert werden. Um die Balance zwischen Zeit und Geld besser zu verstehen, untersuchen Sie im Folgenden das Problem für verschiedene k.

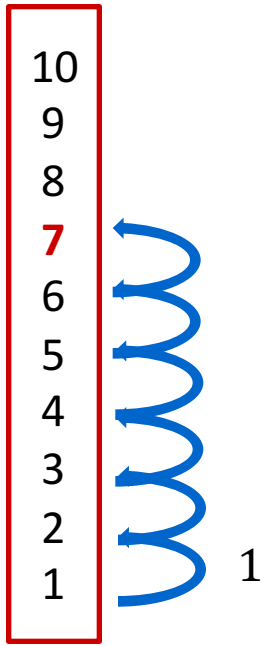
Geben Sie für jeden der folgenden Fälle ein Verfahren an, dass die Anzahl der Testversuche im Worst-Case minimiert und höchstens k Smartphones verwendet:

a) $k = 1$ b) $k = \infty$ *c)* $k = 2$



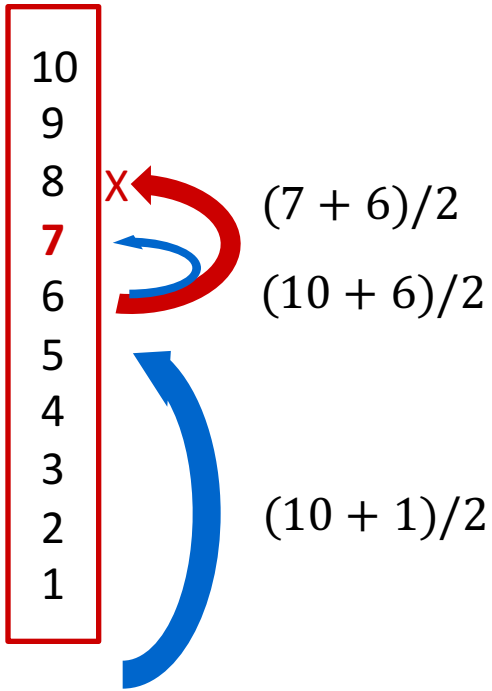
Übungsaufgabe: Suche

Sequentiell



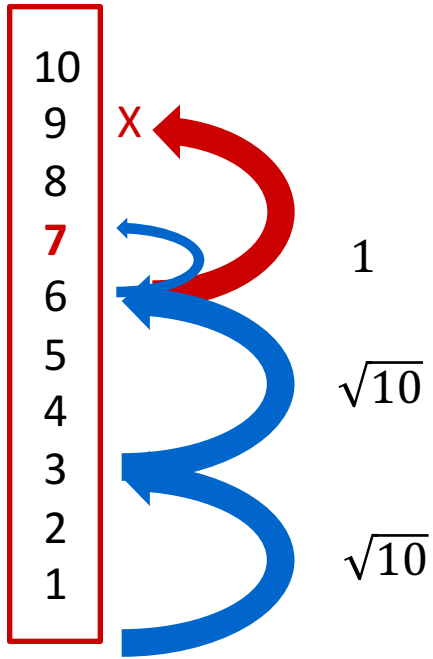
Kosten: $2\left(\sum_{i=1}^{6} i\right) + 7$

Binäre Suche



Kosten: $2(5 + 8) + 7$

Sprung Suche



Kosten: $2(3 + 6 + 9) + 7$