Informatik II Prüfungsvorbereitungskurs

Tag 4, 9.6.2017

Giuseppe Accaputo

g@accaputo.ch

Aufbau des PVK

- **Tag 1**: Java Teil 1
- **Tag 2**: Java Teil 2
- **Tag 3**: Algorithmen & Komplexität
- **Tag 4**: Dynamische Datenstrukturen, Hashing, Suchbäume, Graphen, Datenbanksysteme

Programm für heute

- Repetition
- Dynamische Datenstrukturen
- Hashing
- Suchbäume
- Graphen
- Datenbanksysteme

Repetition Tag 3

Beispiel Klassen

```
class Person{
  String name;
  int alter;
  private String ahvnr;

  public Person(String name, int alter){
    this.name = name;
    this.alter = alter;
  }
  public void setAHVNr(String ahvnr){
    this.ahvnr = ahvnr;
  }
  public String getAHVNr(){
    if(this.ahvnr == null) return "Keine AHV
Nr. Vorhanden!";
    return ahvnr;
  }
}
```

```
…void main(String[] args) {
  Person p1 = new Person("Giu", 30);
  Person p2 = new Person("Hans", 45);
  Person p3 = new Person("Karl", 17);

  p1.setAHVNr("124.187");
  …
}
```

Beispiel Klassen

```java
class ListNode{
  int value;
  ListNode next;

  public ListNode(int value, ListNode next){
    this.value = value;
    this.next = next;
  }
}
```

Beispiel Klassen

```
class SlidingWindow {
    int buf[]; // data buffer
    int size; // how many elements of the buffer are used
    int position; // current insertion position of the buffer

    public SlidingWindow(int size) {
        buf = new int[size];
        position = 0;
        size = 0;
    }

    // post: value added to internal buffer
    public void Put(int value) {
        buf[position] = value;
        position = (position + 1) % buf.length;
        if (size < buf.length) {
            size++;
        }
    }
```

Asymptotische Komplexität

$\mathcal{O}(1)$	beschränkt	Array-Zugriff
$\mathcal{O}(\log\log n)$	doppelt logarithmisch	Binäre sortierte Suche interpoliert
$\mathcal{O}(\log n)$	logarithmisch	Binäre sortierte Suche
$\mathcal{O}(\sqrt{n})$	wie die Wurzelfunktion	Primzahltest (naiv)
$\mathcal{O}(n)$	linear	Unsortierte naive Suche
$\mathcal{O}(n\log n)$	superlinear / loglinear	Gute Sortieralgorithmen
$\mathcal{O}(n^2)$	quadratisch	Einfache Sortieralgorithmen
$\mathcal{O}(n^c)$	polynomial	Matrixmultiplikation
$\mathcal{O}(2^n)$	exponentiell	Travelling Salesman Dynamic Programming
$\mathcal{O}(n!)$	faktoriell	Travelling Salesman naiv

Quelle: Informatik II Vorlesung

Suchen

- **Lineare Suche durch unsortiertes Array**
  - best-case-Laufzeit: $O(1)$
  - worst-case-Laufzeit: $O(n)$
- **Binäre Suche durch sortiertes Array**
  - best-case-Laufzeit: $O(1)$
  - worst-case-Laufzeit: $O(\log(n))$

Auswahl

- **`Partition(A[l..r], p)`**
  - best-case-Laufzeit: $O(n)$
  - worst-case-Laufzeit: $O(n^2)$
- **`Quickselect(A[l..r], i)`**
  - best-case-Laufzeit: $O(n)$
  - worst-case-Laufzeit: $O(n^2)$

Sortieren

- **Selection Sort**
  - worst-case Anzahl Vergleiche: $\Theta(n^2)$
  - best-case Anzahl Vergleiche: $\Theta(n^2)$
  - worst-case Anzahl Vertauschungen: $\Theta(n)$
- **Insertion Sort**
  - worst-case Anzahl Vergleiche: $O(n^2)$
  - best-case Anzahl Vergleiche: $O(n)$
  - worst-case Anzahl Vertauschungen: $O(n^2)$

Sortieren

- **Quicksort**
  - worst-case Anzahl Vergleiche: $\Theta(n^2)$
    - Pivot = Minimum oder Maximum
  - best-case Anzahl Vergleiche: $O(n \log(n))$
    - Pivot = Median
  - Bemerkung: Der randomisierte Quicksort (Pivot = Zufällig) benötigt im Mittel $O(n \log(n))$ Vergleiche

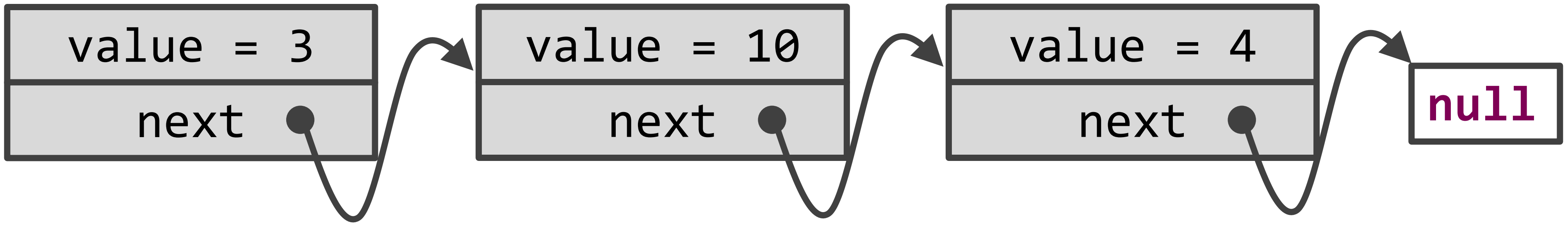
Dynamische Datenstrukturen

Verkettete Liste

```java
class ListNode{
  int value;
  ListNode next;

  public ListNode(int value, ListNode next){
    this.value = value;
    this.next = next;
  }
}
```

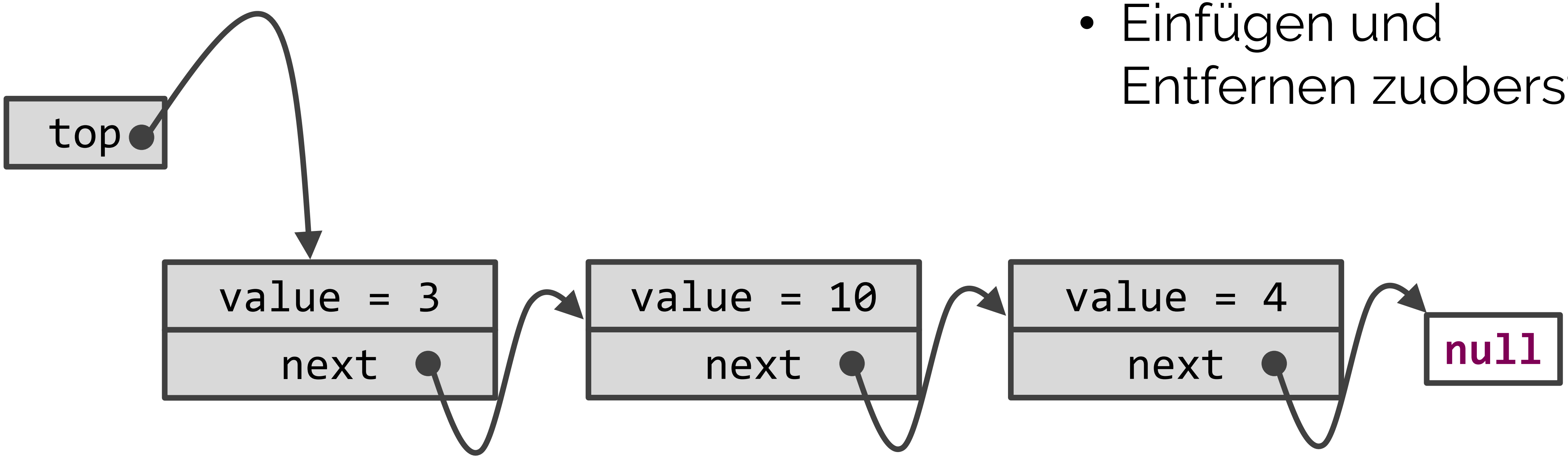
- Kein wahlfreier Zugriff
- Einfügen und Löschen ist einfach



Stack (Stapel)

```
class Stack{
  private ListNode top;
  …
}
```

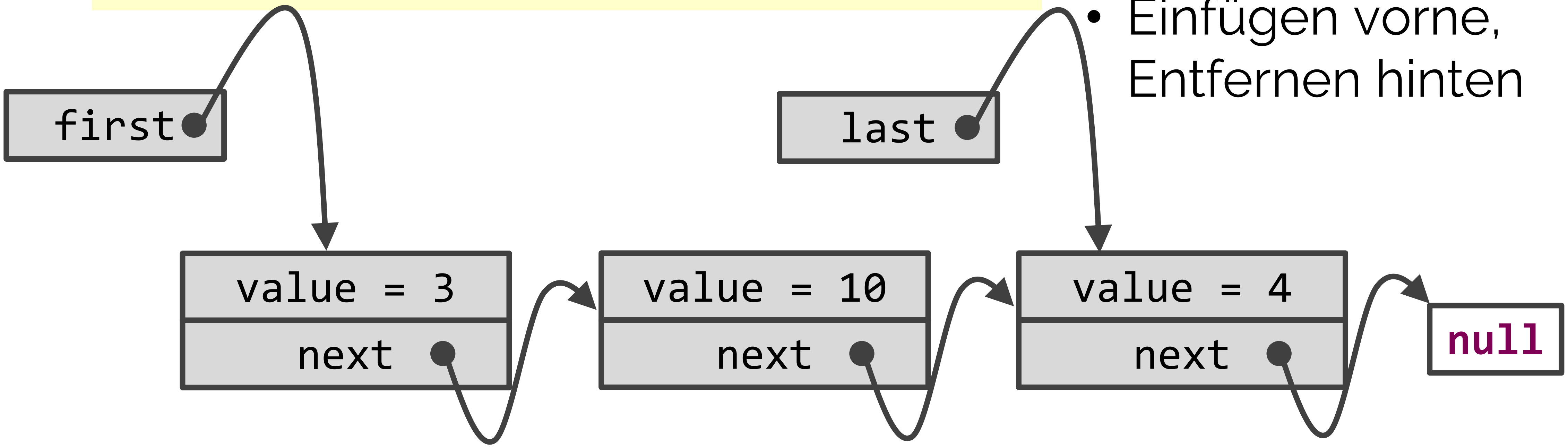
- LIFO:
  Last-In-First-Out
- Einfügen und
  Entfernen zuoberst



Queue (Schlange)

```java
class Queue{
  private ListNode first;
  private ListNode last;
  …
}
```

- FIFO:
  First-In-First-Out
- Einfügen vorne,
  Entfernen hinten

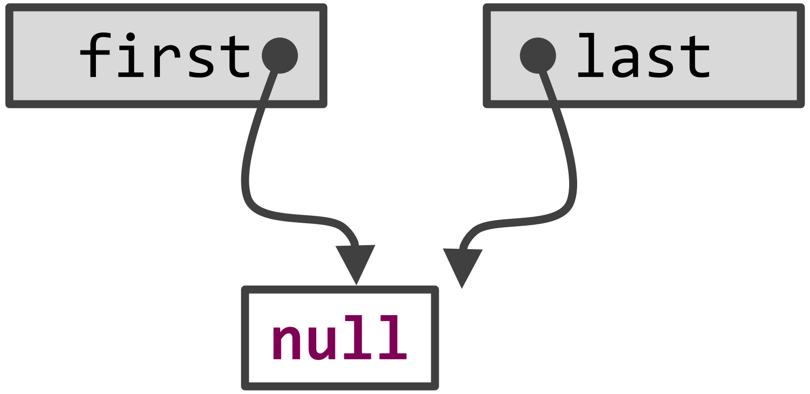


Queue: Operationen

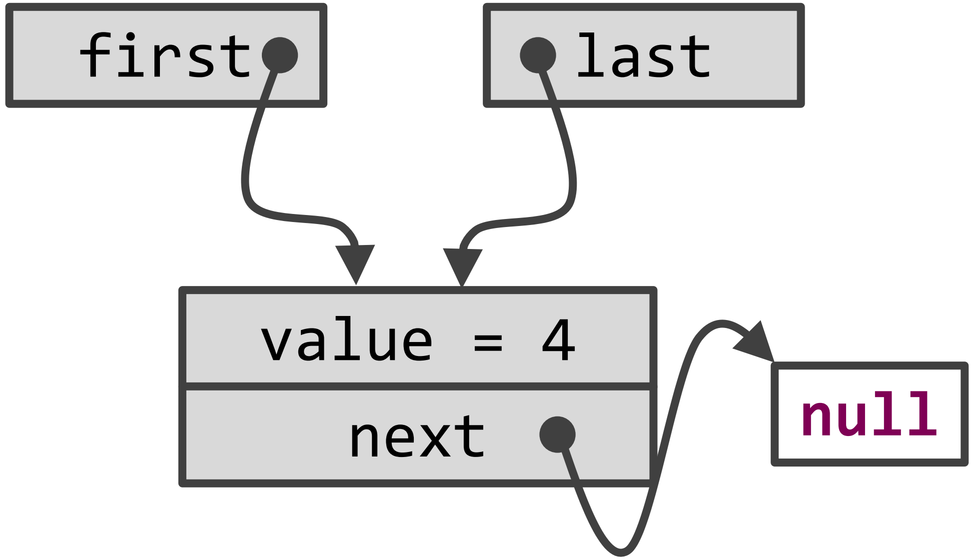
- `enqueue(x)`: fügt `x` am Anfang der Schlange ein
- `dequeue()`: Entfernt `x` vom Ende der Schlange und gibt `x` zurück (`null` sonst)
  - **Wichtig**: Bei FIFO Einfügen an eine Seite, und Löschen von der anderen Seite. Am Anfang löschen und Ende einfügen daher auch möglich!
- `head()`: Gibt das Objekt am Beginn der Schlange zurück (`null` sonst)
- `empty()`: liefert `true` wenn Queue leer, sonst `false`

Zustände einer Queue

Leere Queue

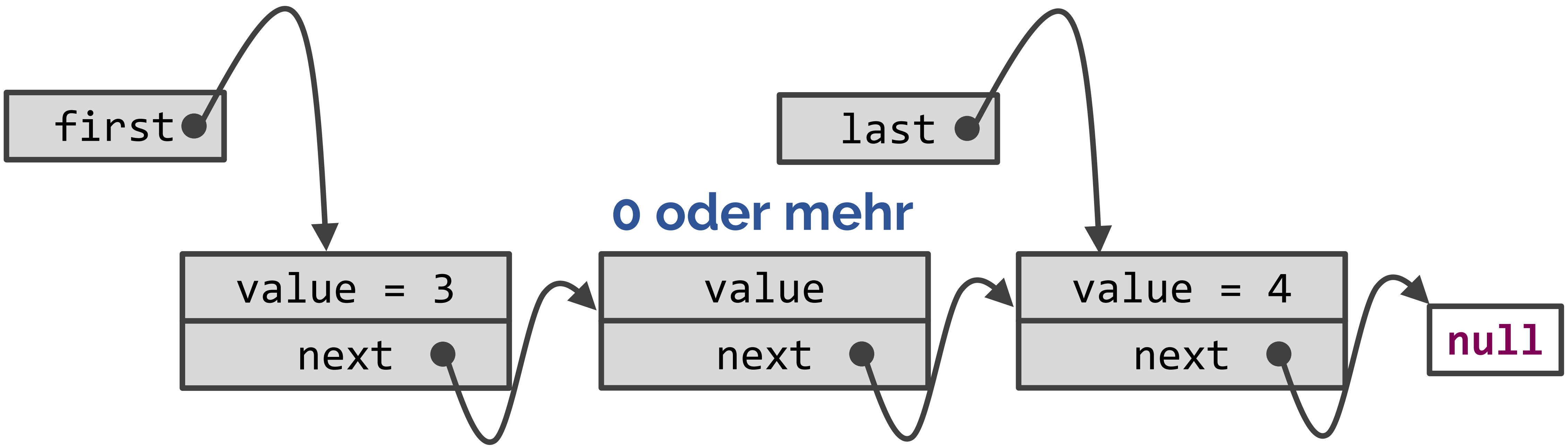


Queue mit nur einem Element



Zustände einer Queue

Queue mit zwei oder mehreren Elementen



Queue: Implementation der Operationen

```java
class Queue{
  ListNode first;
  ListNode last;

  public Queue(){
    first = null;
    last = first; //Leere Liste
  }

  public void insert(int n){…}
  public int remove(){…}
  public ListNode head(){…}
  public boolean empty(){…}
}
```

```java
class ListNode{
  int value;
  ListNode next;

  public ListNode(int value, ListNode next){
    this.value = value;
    this.next = next;
  }
}
```

→ Hellraumprojektor :)

Queue: Durchiterieren

- Z.B. um Elemente auszugeben oder auch um nach einem Element zu suchen

```
public void iterate(){
  ListNode l = first;

  while(l != null){
    …
    l = l.next;
  }
}
```

Prüfungsaufgaben

- **Prüfung 01.2015**: Aufgabe 4
- **Prüfung 08.2016**: Aufgabe 5
- Weitere Aufgaben
  - Prüfung 08.2014: Aufgabe 8a & 8b
  - Prüfung 08.2015: Aufgabe 4b
  - Prüfung 02.2016: Aufgabe 5a & 5b

Hashing

Hashing

Hashfunktion h: Abbildung aus der Menge der Schlüssel $\mathcal{K}$ auf die Indexmenge $\{0, 1, \ldots, m-1\}$ eines Arrays (*Hashtabelle*).

$$h : \mathcal{K} \to \{0, 1, \ldots, m-1\}.$$

Meist $|\mathcal{K}| \gg m$. Es gibt also $k_1, k_2 \in \mathcal{K}$ mit $h(k_1) = h(k_2)$ (*Kollision*).

Eine Hashfunktion sollte die Menge der Schlüssel möglichst gleichmässig auf die Positionen der Hashtabelle verteilen.

Quelle: Informatik II Vorlesung

Hashing

$$h_{b,m}(s) = \left(\sum_{i=0}^{l-1} s_i \cdot b^i \right) \bmod m$$

```
int ComputeHash(int m, String s) {
  int sum = 0;
  int b = 1;
  for (int k = 0; k<s.length(); ++k){
    sum = (sum + s.charAt(k) * b) % m;
    b = (b * 31) % m;
  }
  return sum;
}
```

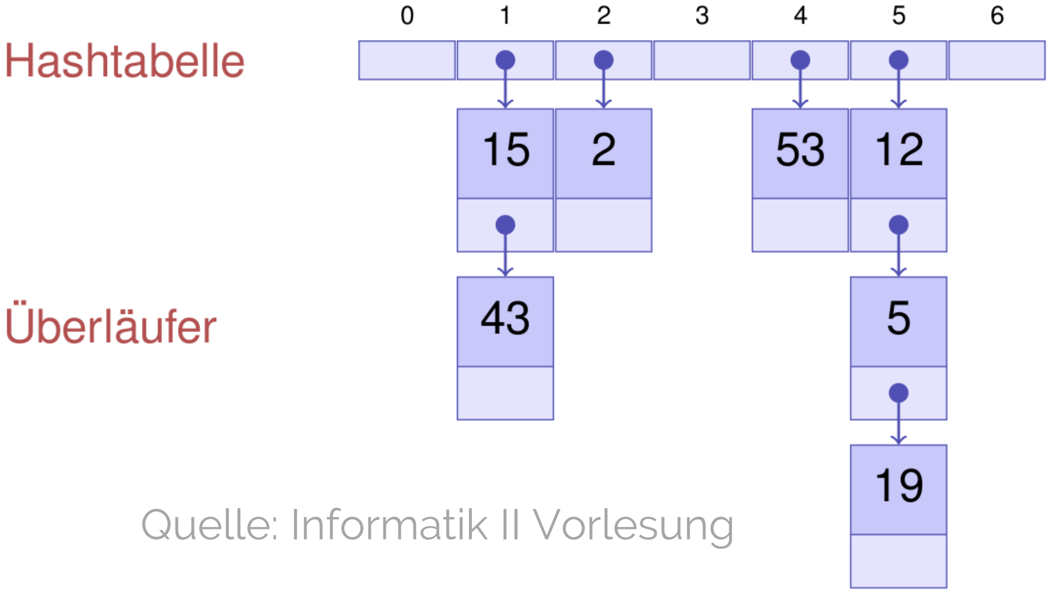
Quelle: Informatik II Vorlesung

Hashing: Behandlung von Kollisionen

Beispiel $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Schlüssel 12 , 53 , 5 , 15 , 2 , 19 , 43

Direkte Verkettung der Überläufer



Quelle: Informatik II Vorlesung

Hashing: Behandlung von Kollisionen

```java
public class ListNode {
    String key;
    int value;
    ListNode next;

    ListNode (String k, int val, ListNode nxt) {
        key = k;
        value = val;
        next = nxt;
    }
}
```

Quelle: Informatik II Vorlesung

Hashing: Behandlung von Kollisionen

```
class HashTable {
    ListNode[] data;
    int m;

    HashTable(int size) {
        m = size;
        data = new ListNode[size];
    }
```

Quelle: Informatik II Vorlesung

```
    //pre: String of length > 0
    //post: returns a string which
    int computeHash(String s) {
```

Hashing: Behandlung von Kollisionen

```
//pre: hash table contains the key
//post: return the value stored with the key
int get(String key) {
    assert(contains(key));
    int h = computeHash(key);
    ListNode n = data[h];
    while (n != null && !key.equals(n.key)) {
        n = n.next;
    }
    return n.value;
}
```

Quelle: Informatik II Vorlesung

Offene Hashverfahren

$s(j,k) = j \Rightarrow$
$S(k) = (h(k) \bmod m, (h(k)-1) \bmod m, \ldots, (h(k)+1) \bmod m)$

Beispiel $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.
Schlüssel 12 , 53 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2	5	53	12	

Quelle: Informatik II Vorlesung

Offene Hashverfahren

$$s(j,k) = \lceil j/2 \rceil^2 (-1)^j$$
$$S(k) = (h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Beispiel $m = 7, \mathcal{K} = \{0, \dots, 500\}, h(k) = k \bmod m$.

Schlüssel 12 , 53 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2		53	12	5

Quelle: Informatik II Vorlesung

Offene Hashverfahren

Zwei Hashfunktionen $h(k)$ und $h'(k)$. $s(j,k) = j \cdot h'(k)$.

$S(k) = (h(k) - h'(k), h(k) - 2h'(k), \ldots, h(k) - (m-1)h'(k)) \mod m$

Beispiel:

$m = 7, \mathcal{K} = \{0, \ldots, 500\}, h(k) = k \mod 7, h'(k) = 1 + k \mod 5.$

Schlüssel 12 , 53 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2	5	53	12	

Quelle: Informatik II Vorlesung

Aufgabe Hashing

- Übung 8, Aufgabe 2b

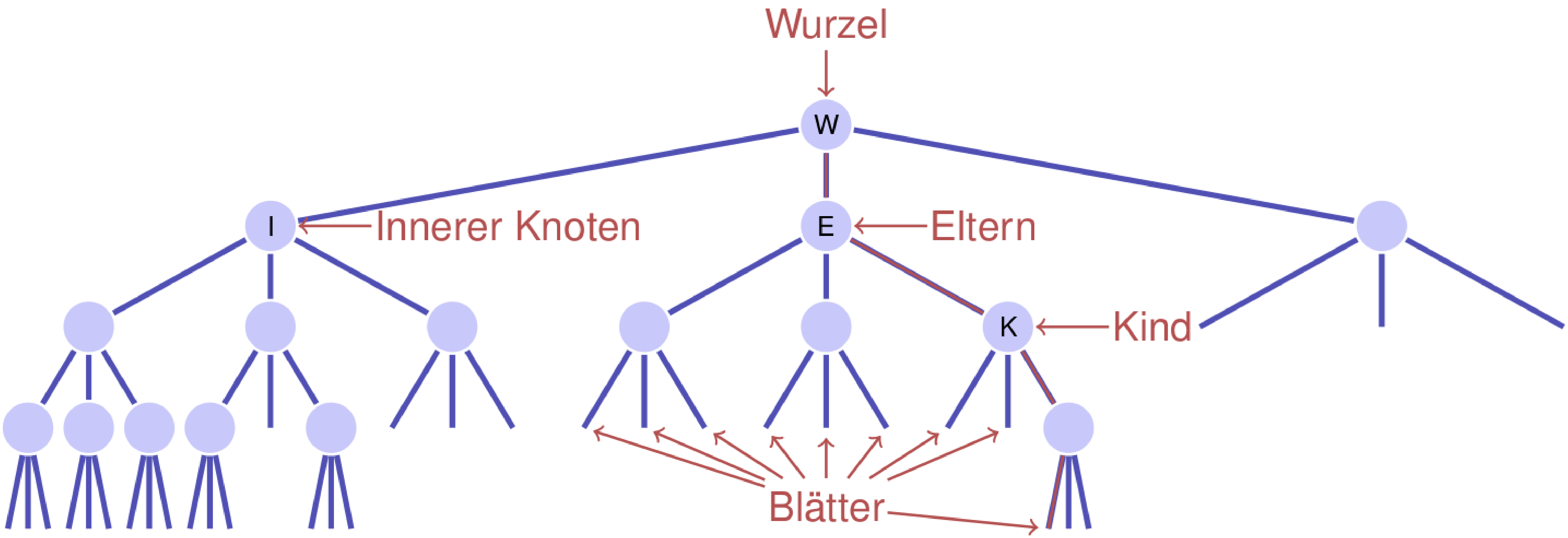
Natürliche Suchbäume

Natürliche Suchbäume

Bäume sind

- Verallgemeinerte Listen: Knoten können mehrere Nachfolger haben
- Spezielle Graphen: Graphen bestehen aus Knoten und Kanten

Nomenklatur



Quelle: Informatik II Vorlesung

Binäre Bäume

Ein binärer Baum ist entweder

- Ein Blatt, d.h. ein leerer Baum
- Ein innerer Knoten mit zwei Bäumen als linken und rechten Nachfolger

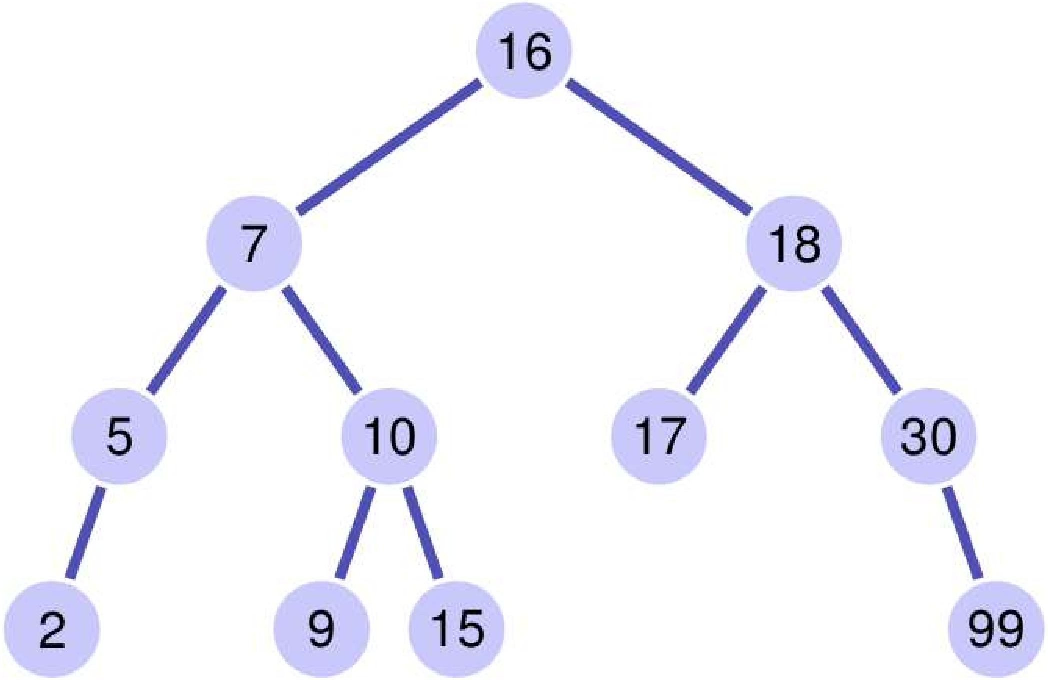
```java
public class SearchNode {
  int key;        // Schluessel
  SearchNode left;     // linker Teilbaum
  SearchNode right;    // rechter Teilbaum

  // Konstruktor: Knoten ohne Nachfolger
  SearchNode(int k){
    key = k;
    left = right = null;
  }
}
```

Quelle: Informatik II Vorlesung

Binärer Suchbaum

- Jeder Knoten speichert einen Schlüssel
- `Schlüssel` im linken Teilbaum von **v** sind kleiner als der Schlüssel von **v**
- Schlüssel im rechten Teilbaum von **v** sind grösser als der Schlüssel von **v**



Quelle: Informatik II Vorlesung

Binärer Suchbaum: Suchen

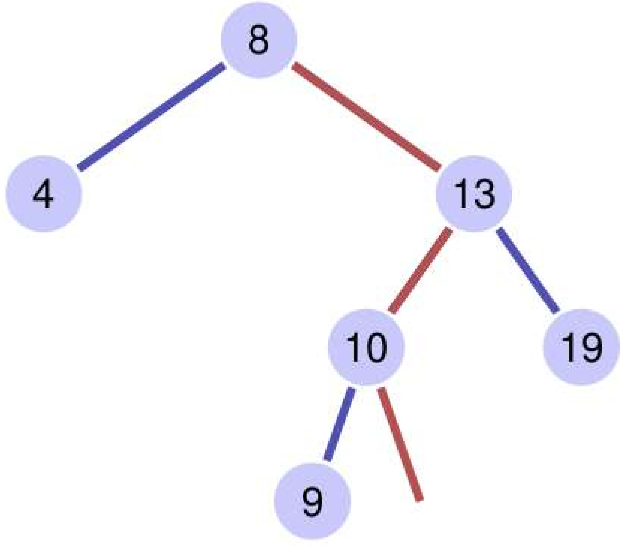
Input : Binärer Suchbaum mit Wurzel r,
Schlüssel k

Output : Knoten v mit $v.\text{key} = k$ oder **null**

$v \leftarrow r$

while $v \neq$ **null do**
  if $k = v.\text{key}$ **then**
    return v
  else if $k < v.\text{key}$ **then**
    $v \leftarrow v.\text{left}$
  else
    $v \leftarrow v.\text{right}$

return null



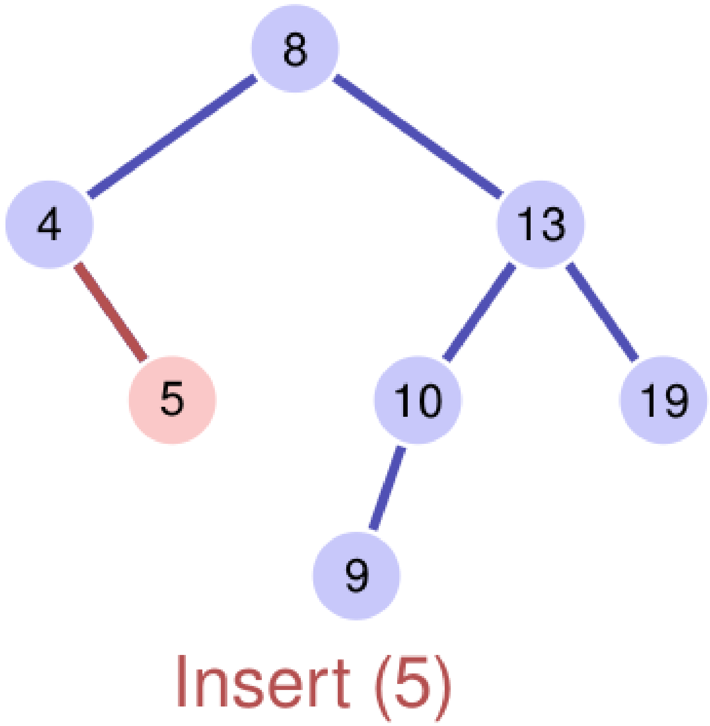
Search (12) $\rightarrow$ **null**

Quelle: Informatik II Vorlesung

Binärer Suchbaum: Einfügen

Einfügen des Schlüssels k

- Suche nach k.
- Wenn erfolgreich: Fehlerausgabe
- Wenn erfolglos: Einfügen des Schlüssels am erreichten Blatt.
- Implementation: der Teufel steckt im Detail



Quelle: Informatik II Vorlesung

Binärer Suchbaum: Knoten entfernen

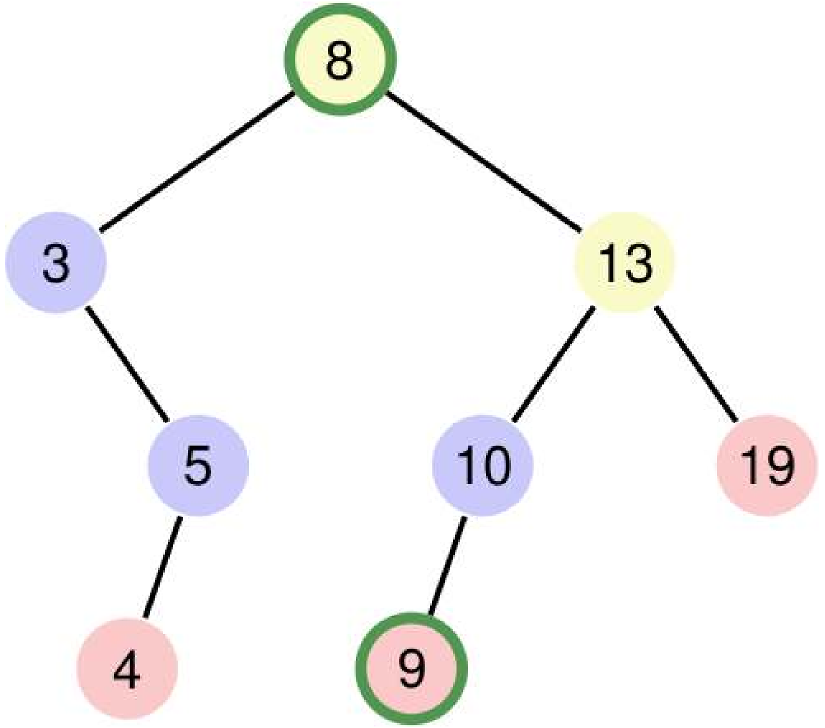
- Knoten hat keine Kinder: Knoten durch Blatt ersetzen
- Knoten hat ein Kind: Knoten durch das einzige Kind ersetzen
- Knoten hat zwei Kinder: Knoten durch symmetrischen Nachfolger (oder Vorgänger) ersetzen

Binärer Suchbaum: Knoten entfernen

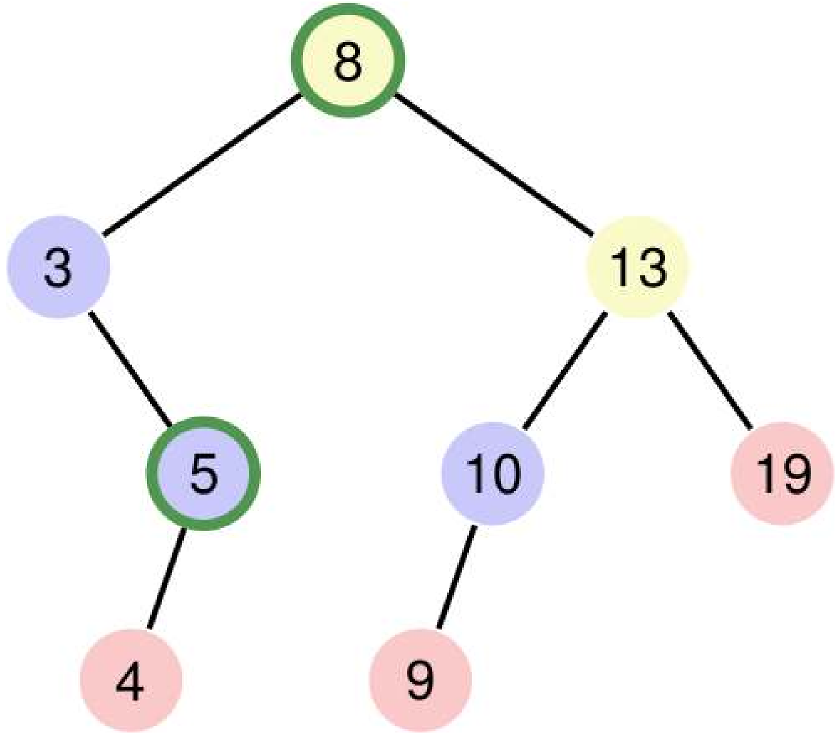
- **Symmetrischer Nachfolger**: kleinster Schlüssel im rechten Teilbaum des zu entfernenden Knoten v
- **Symmetrischer Vorgänger**: grösster Schlüssel im linken Teilbaum des zu entfernenden Knoten v

Binärer Suchbaum: Knoten entfernen

- **Symmetrischer Nachfolger :**



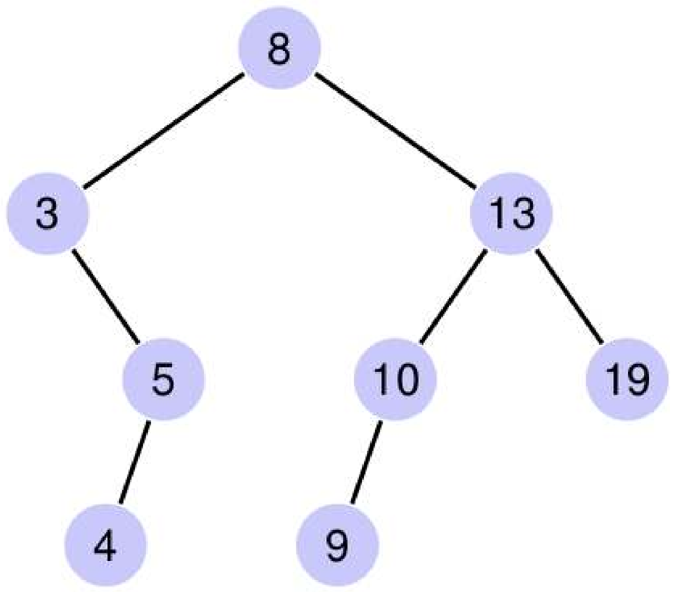
- **Symmetrischer Vorgänger:**



Quelle: Informatik II Vorlesung

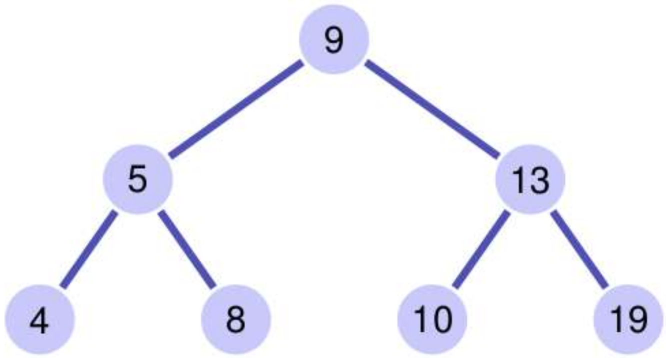
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19

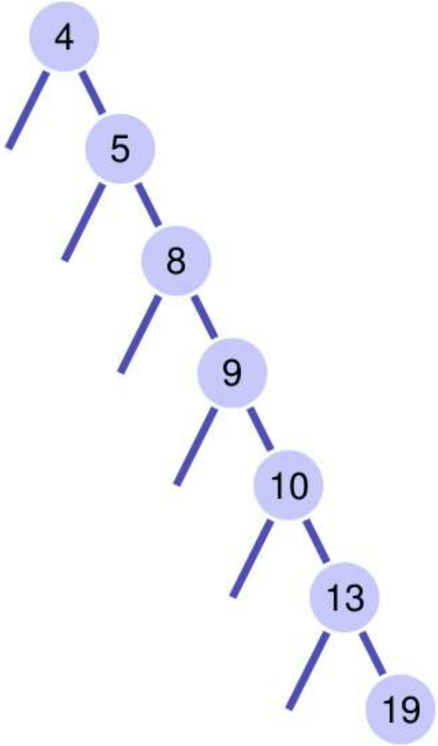


Quelle: Informatik II Vorlesung

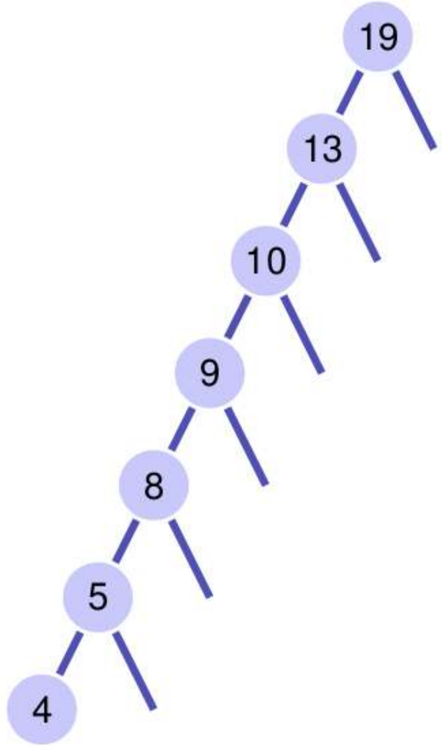
Degenerierte Bäume



Insert 9,5,13,4,8,10,19
bestmöglich
balanciert



Insert 4,5,8,9,10,13,19
Lineare Liste



Insert 19,13,10,9,8,5,4
Lineare Liste

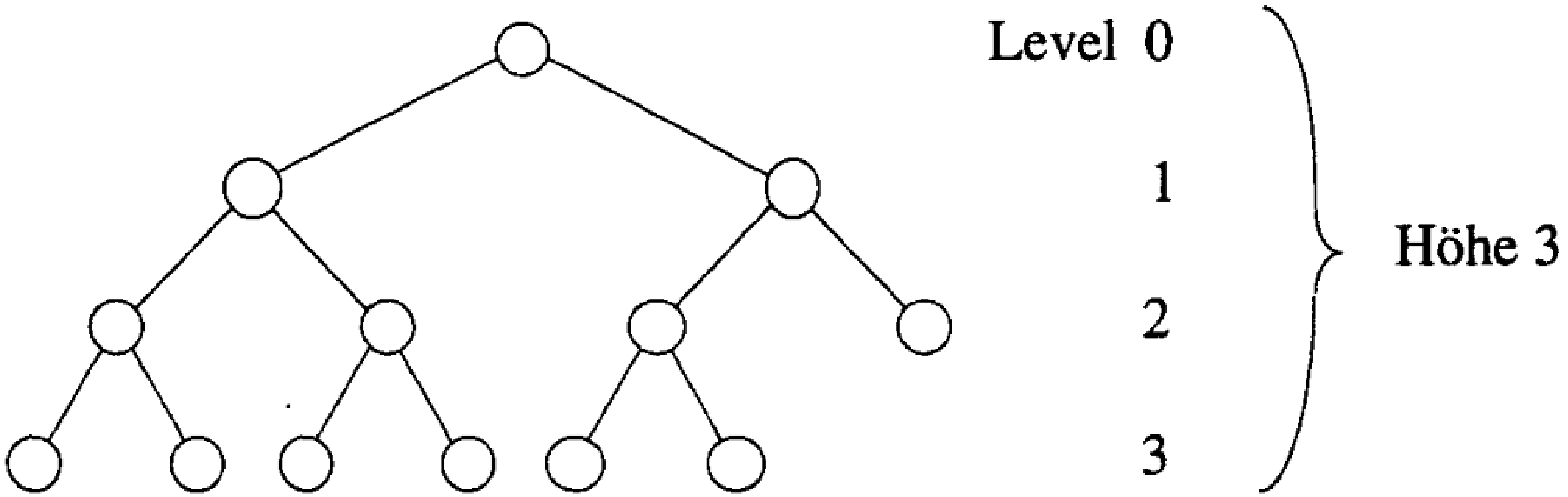
Quelle: Informatik II Vorlesung

Binärer Suchbaum: Laufzeiten

- $h(T)$ = Höhe des Baums T mit Wurzel r
  - max. Pfadlänge Wurzel bis Blatt (Blatt exklusive)
- Suchen:
  - **worst-case-Laufzeit: $\boldsymbol{O\big(h(T)\big)}$**
- Entfernen:
  - **worst-case-Laufzeit: $\boldsymbol{O\big(h(T)\big)}$**
- Suche des symmetrischen Nachfolgers:
  - **worst-case-Laufzeit: $\boldsymbol{O\big(h(T)\big)}$**

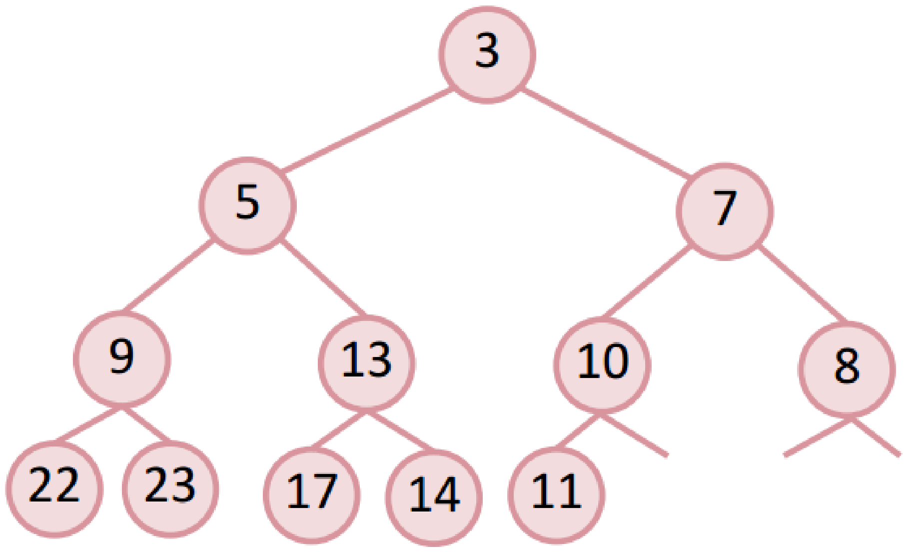
Balancierte Bäume

- Garantiere, dass Binärbaum mit n Knoten stets eine Höhe von $O(log_2(n))$



Bildquelle: «Datenstrukturen und Algorithmen» (Güting & Dieker)

Heap

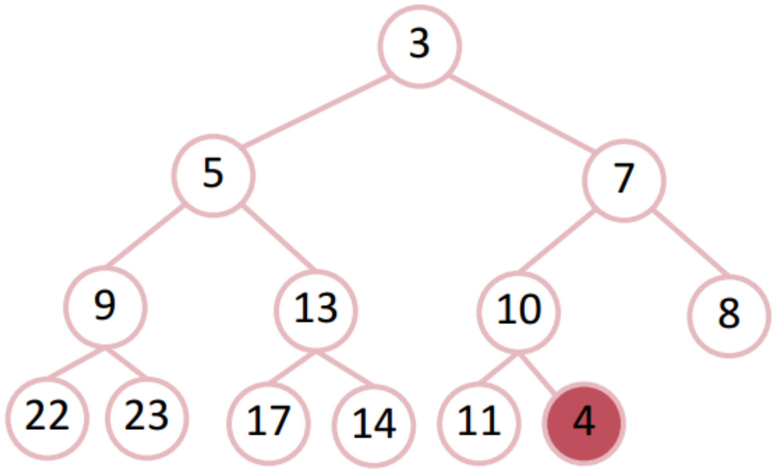


Quelle: Informatik II Vorlesung FS 2016

- Heap ist ein Binärbaum
- Min-Heap-Eigenschaft: Schlüssel eines Kindes ist immer grösser als der des Vaters
  - Max-Heap: Kinder-Schlüssel immer kleiner als Vater-Schlüssel
- Heap hat nur Lücken in der letzten Ebene; müssen alle rechts liegen

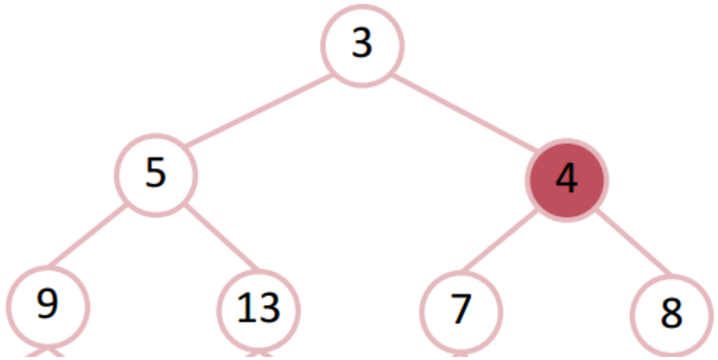
Heap: Knoten einfügen

1. Füge neuen Knoten an ersten freien Stelle (verletzt Heap-Eigenschaft)



2. Stelle Heap-Eigenschaft wieder her durch sukzessives Aufsteigen des Knotens

$$O(\log(n))$$



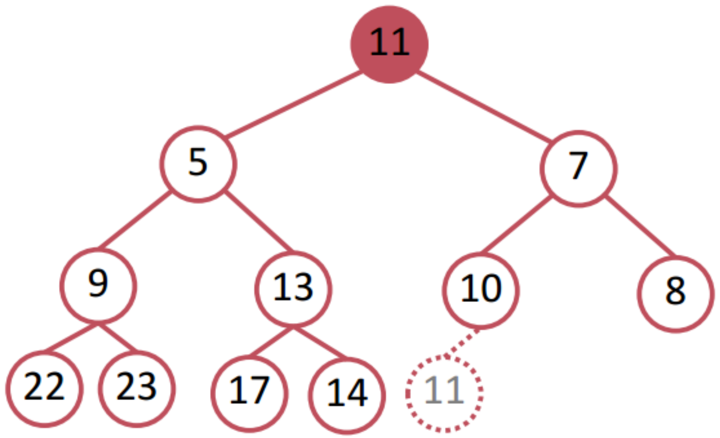
Quelle: Informatik II Vorlesung FS 2016

Heap: Wurzelknoten

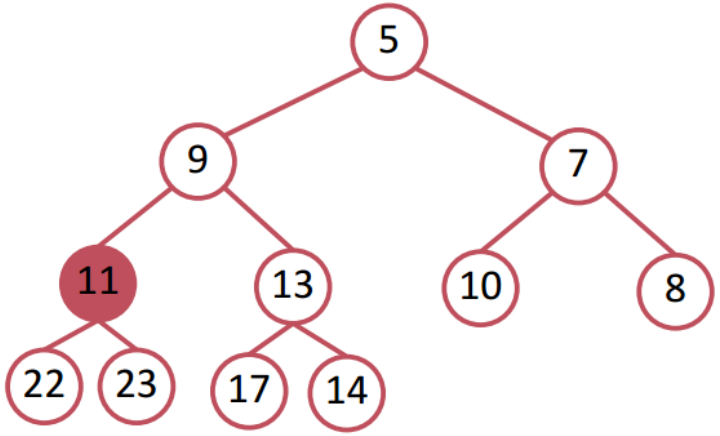
- Bei min-Heap ist der Wurzelknoten das kleinste Element
- Bei max-Heap ist der Wurzelknoten das grösste Element

Heap: Wurzelknoten entfernen

1. Ersetze Wurzel durch den letzten Knoten



2. Lasse Wurzel nach unten sinken, um Heap-Eigenschaft wiederherzustellen



$O(\log(n))$

Quelle: Informatik II Vorlesung FS 2016

Heap: Laufzeiten

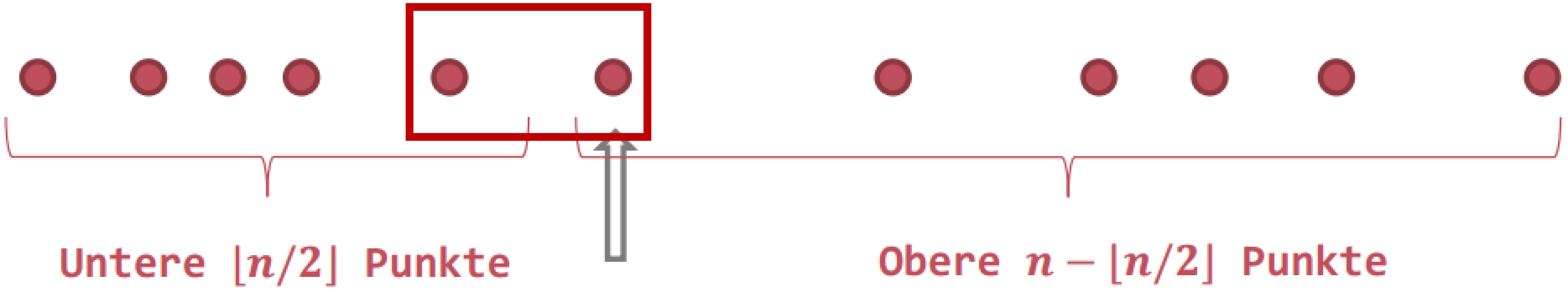
- $h(T) = \log_2(n)$ = Höhe des Heaps
- **Suchen**:
  - **worst-case-Laufzeit: $\boldsymbol{O(n)}$ (!!!)**
- Einfügen:
  - **worst-case-Laufzeit: $\boldsymbol{O(\log(n))}$**
- Wurzel entfernen:
  - **worst-case-Laufzeit: $\boldsymbol{O(\log(n))}$**
- Wurzel **nur** auslesen (Max. oder Min.):
  - **worst-case-Laufzeit: $\boldsymbol{O(1)}$**

Prüfung 02.2016 Aufgaben 7a & 7b

- Hellraumprojektor

Online Median

- Verwende das schnelle Auslesen, Extrahieren und Einfügen vom Minimum (bzw. Maximum) im Heap für den Online Median
- Median bildet sich aus Minimum der oberen Hälfte der Daten und / oder Maximum der unteren Hälfte



Online Median: Algorithmus

- Verwende Max-Heap um untere Hälfte und Min-Heap um obere Hälfte der Elemente zu speichern
- Füge Wert `v` in
  - Max-Heap, falls kleiner oder gleich Wurzelknoten ist
  - Min-Heap sonst
- Rebalancieren der beiden Heaps
  - Falls Max-Heap mehr als Hälfte der Elemente hat, dann platziere `max(Max-Heap)` (Wurzel) in Min-Heap
  - Falls Max-Heap weniger als Hälfte der Elemente hat, dann platziere `min(Min-Heap)` (Wurzel) in Max-Heap

$O(\log(n))$

Online Median: Berechnung Median

- Wenn Anzahl Elemente ungerade ist, dann ist Median gleich der `min(Min-Heap)`
- Wenn Anzahl Elemente gerade ist, dann ist der Median definiert als `[max(Max-Heap) + min(Min-Heap)]/2`

$$O(1)$$

Online Median

```
public void Insert(double value) {
    if (minHeap.size() == 0 || value > minHeap.GetRoot()) {
        minHeap.Insert(value);
    } else {
        maxHeap.Insert(value);
    }
    n++;
    if (maxHeap.size() < n/2) {
        maxHeap.Insert(minHeap.ExtractRoot());
    } else if (maxHeap.size() > n/2) {
        minHeap.Insert(maxHeap.ExtractRoot());
    }
}
```

Online Median

```
public double GetMedian() {
    if (n%2 == 0) {
        return (minHeap.GetRoot() + maxHeap.GetRoot()) / 2;
    } else {
        return minHeap.GetRoot();
    }
}
```

Prüfung 08.2015 Aufgabe 7c

- Hellraumprojektor

Graphen

Notation

- Gerichteter Graph $\mathrm{G}(V, E)$ besteht aus einer Menge von Knoten V und einer Menge von Kanten E
- **Gewichteter Graph** ist ein Graph $\mathrm{G}(V, E)$ mit Knotenmenge V und Kantenmenge E, bei dem jeder Kante eine Länge zugeordnet ist

Suche

- Suche nach bestimmten Knoten: Tiefen- und Breitensuch
- Suche nach kürzestem Weg: Algorithmus von Dijkstra

Prüfung 08.2015 Aufgabe 8b

- Hellraumprojektor

Datenbanksysteme

Abfragen

- Fokus auf Abfragen setzen (relationale Algebra und SQL mit Selektion, Projektion, Joins, etc.)

Prüfungsaufgaben

- Prüfung 01.2015 Aufgabe 3
- Prüfung 08.2015 Aufgabe 9
- Prüfung 02.2016 Aufgabe 9

Vorbereitungstipps

- In Gruppen lernen kann vorteilhaft sein
  - Bei Fragen ist gleich jemand da
  - Fragen beantworten hilft Verständnis
- Unbedingt alle alten Prüfungen lösen (**Wichtig**)
  - Eine Prüfung verwenden, um Prüfungssituation zu simulieren (auf Zeit)
- Übungen anschauen und versuchen zu verstehen
- Blick in alte Übungen werfen
  - http://lec.inf.ethz.ch/baug/informatik2/2014/
  - http://lec.inf.ethz.ch/baug/informatik2/2015/
  - http://lec.inf.ethz.ch/baug/informatik2/2016/

Prüfungstipps

- Zeit pro Punkt ausrechnen: Dauer Prüfung / Anz. Punkte
  - Uhr an Prüfung mitnehmen (z.B. Wecker)
  - Gute Abschätzung um zu sehen, ob man zu viel Zeit mit einer Aufgabe verbringt
- Prüfung zuerst durchblättern und schauen, welche Aufgaben einfach zu lösen sind. **Diese dann gleich als erstes lösen!**

Fragen zu Übungen oder alten Prüfungen

- Ich stehe gerne zur Verfügung für Fragen zu den Übungen und alten Prüfungen, falls welche auftauchen sollten während der Vorbereitung:

g@accaputo.ch

- Meine Slides zu den diesjährigen Informatik II Übungen: http://accaputo.ch/hilfsassistenz/informatik-2-d-baug-2017/