Häufige Fehler (1)

Warum funktioniert dieser Iterator für die Ringeliste nicht?

```
class CreateIterator implements Iterator {
  private WordNode p;
  CreateIterator(WordNode head){
    this.p = head;
  }
  public boolean hasNext() {
    return p.next != null;
  }
  public Object next() {
    p = p.next;
    return p;
  }
  public void remove() {
    throw new UnsupportedOperationException();
  }
}
```

Endlosschleife!

Besser:

```
class SortedWordListIterator implements Iterator {
  SortedWordListIterator(WordNode head) {
    this.head = head;
    p = head.next;
  }
  public void remove() {
    throw new UnsupportedOperationException();
  }
  public boolean hasNext() {
    return (p != head);
  }
  public Object next() {
    WordNode q = p;
    p = p.next;
    return q;
  }
  private WordNode head, p;
}
```

Häufige Fehler (2)

```
// Gibt die Liste in der Console aus
public void print() {
  WordNode p = head.next;
  int i = 1;
  while( p!=head ) {
    Out.println(i + " " + p.word + " (" + p.count + " times)");
    i++;
    p = p.next;
  }
}
```

Warum verwenden Sie *toString()* nicht?

```
// Gibt die Liste in der Console aus
public void print() {
  WordNode p = head.next;
  int i = 1;
  while( p!=head ) {
    Out.println(i + " " + p.toString());
    i++;
    p = p.next;
  }
}
```

oder

```
// Gibt die Liste in der Console aus
public void print() {
  WordNode p = head.next;
  int i = 1;
  while( p!=head ) {
    Out.println(i + " " + p);
    i++;
    p = p.next;
  }
}
```

```
class WordNode {
  public String toString()  {
    return String.format("%s (%d times)", word, count);
  }
  ...
}
```

Häufige Fehler (3)

```
// Gibt die Liste in der Console aus
public void print() {
  WordNode p = head.next;
  int i = 1;
  while( p!=head ) {
    Out.println(i + " " + p);
    i++;
    p = p.next;
  }
}
```

Warum verwenden Sie den *Iterator* nicht?

```
public void print() {
  Iterator it = iterator();
  int cnt = 0;
  while(it.hasNext())
    Out.println(++cnt + " " + it.next());
}
```

oder

```
public void print() {
  int cnt = 0;
  for(Object n: this)
    Out.println(++cnt + " " + (WordNode) n);
}
```

Wiederverwendung!

1) Einfügen in Rot-Schwarz Baum (zeichnen)

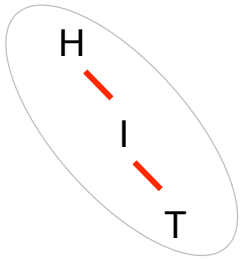
Geben Sie Rot-Schwarz-Bäume an (mit allen Zwischenschritten), die beim Einfügen der Buchstaben HITCHHIKER entstehen.

Einfügen von: HITCHHIKER

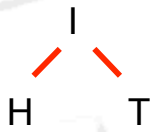
H + I

+ T

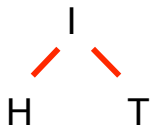
neuer Knoten wird immer über rote Kante angehängt



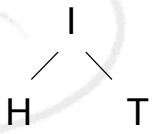
⇒ 1x rotieren

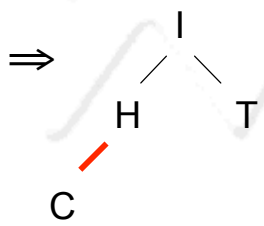


+ C



⇒ splitten

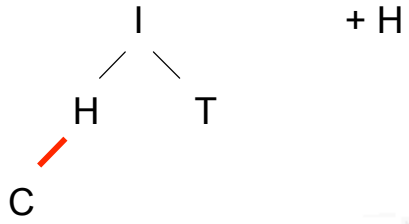




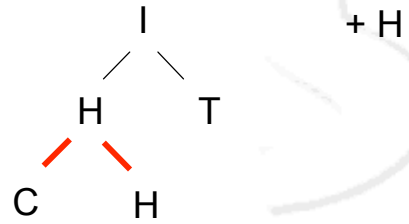
+ H

Rot-Schwarz-Baum

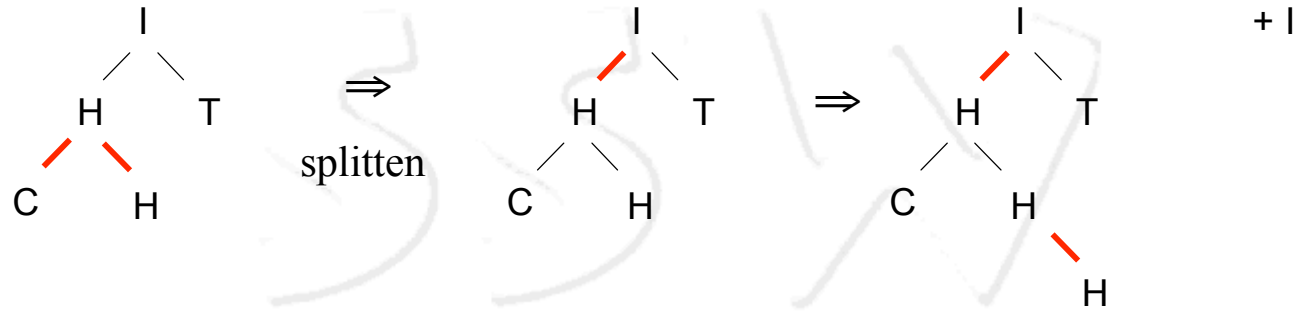
Einfügen von: HITCHHIKER



+ H



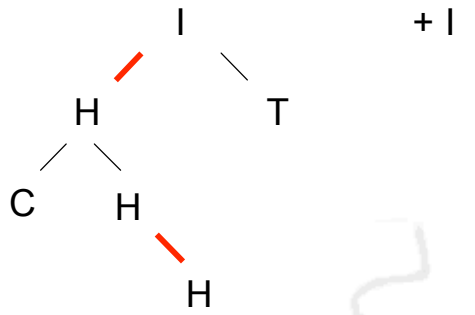
+ H



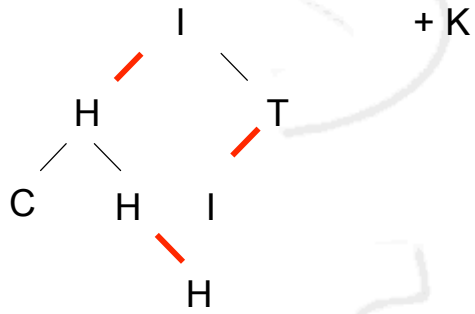
+ I

Rot-Schwarz-Baum

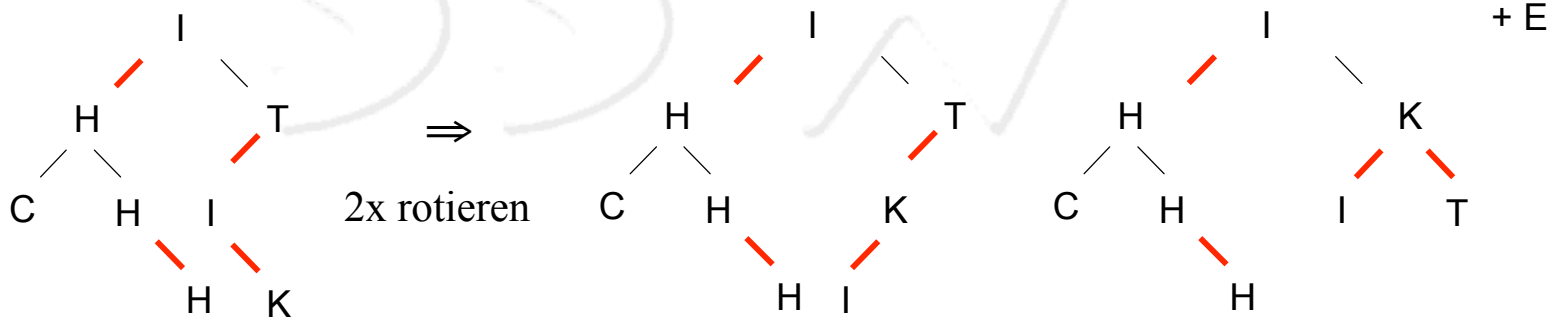
Einfügen von: HITCHHIKER



+ I



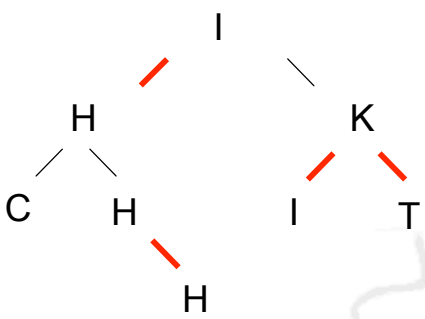
+ K



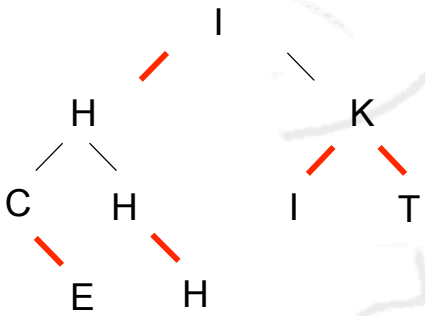
+ E

Rot-Schwarz-Baum

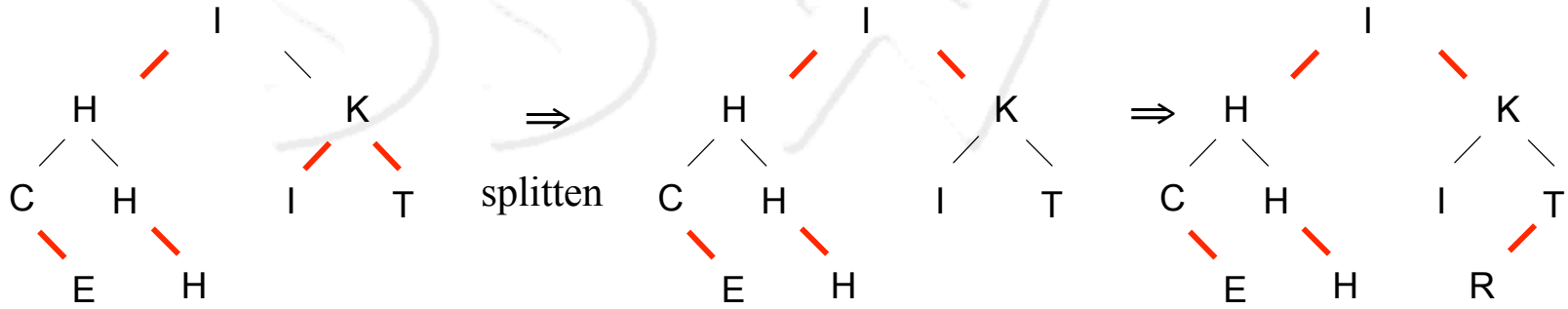
Einfügen von: HITCHHIKER



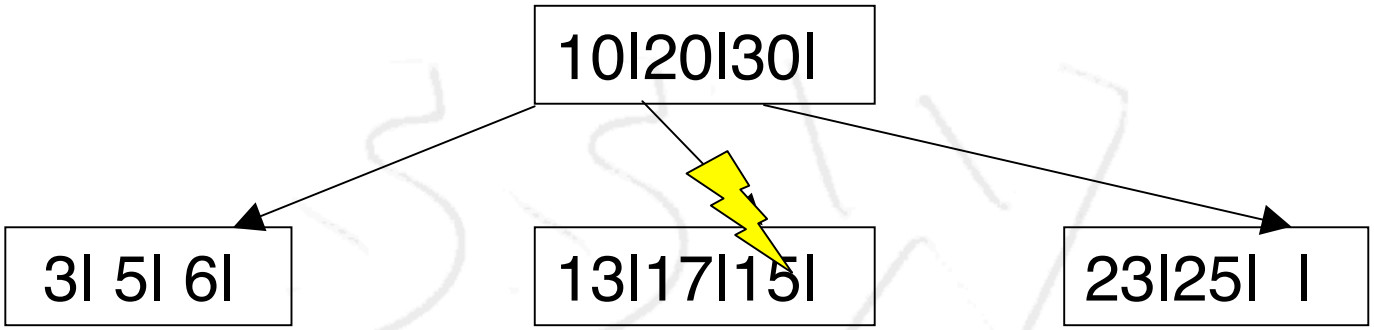
+ E



+ R

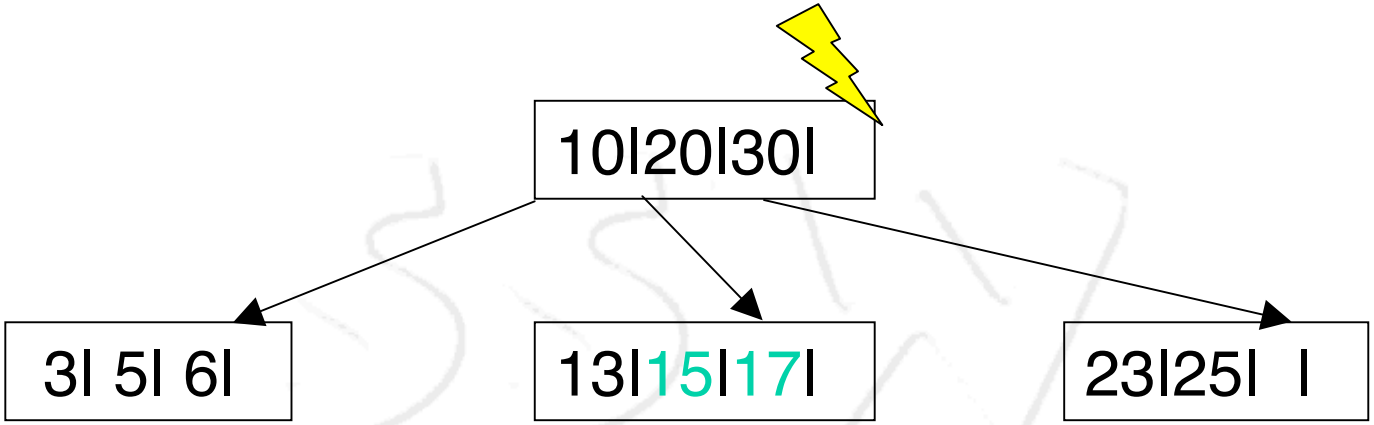


- Ist folgender Mehrwegbaum vom Grad 2?



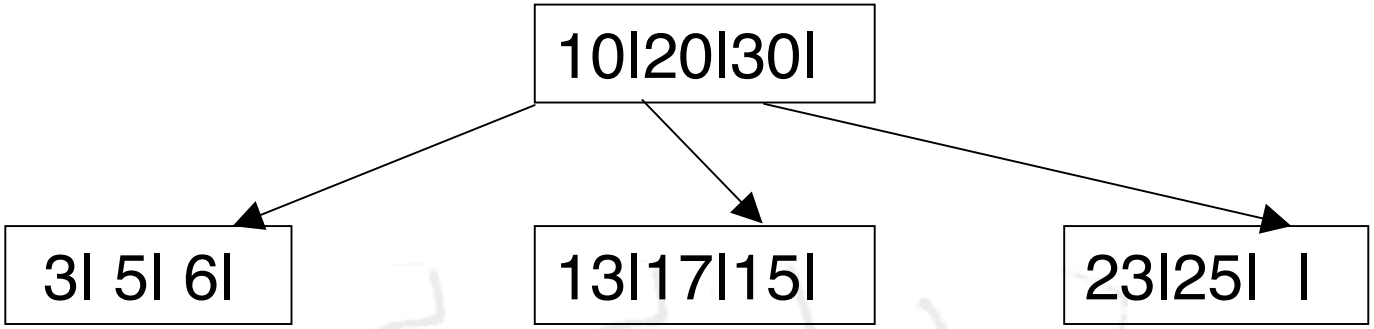
Nein!
Die Schlüssel müssen im jedem Knoten aufsteigend sortiert sein.

- Ist folgender Mehrwegbaum vom Grad 2?

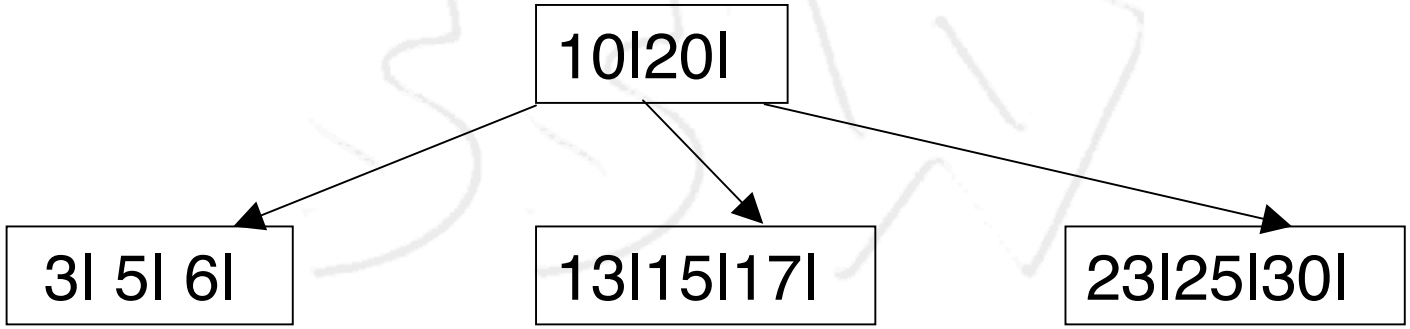


Nein!
Der linke und rechte Teilbaum eines Schlüssels darf nicht leer sein.

B-Baum

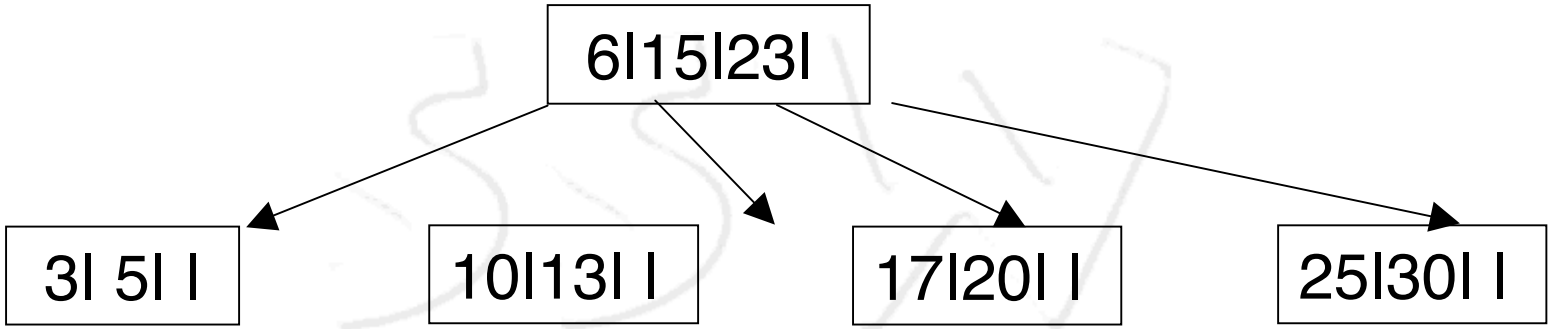


- Gültiger B-Baum mit gleichen Werten:



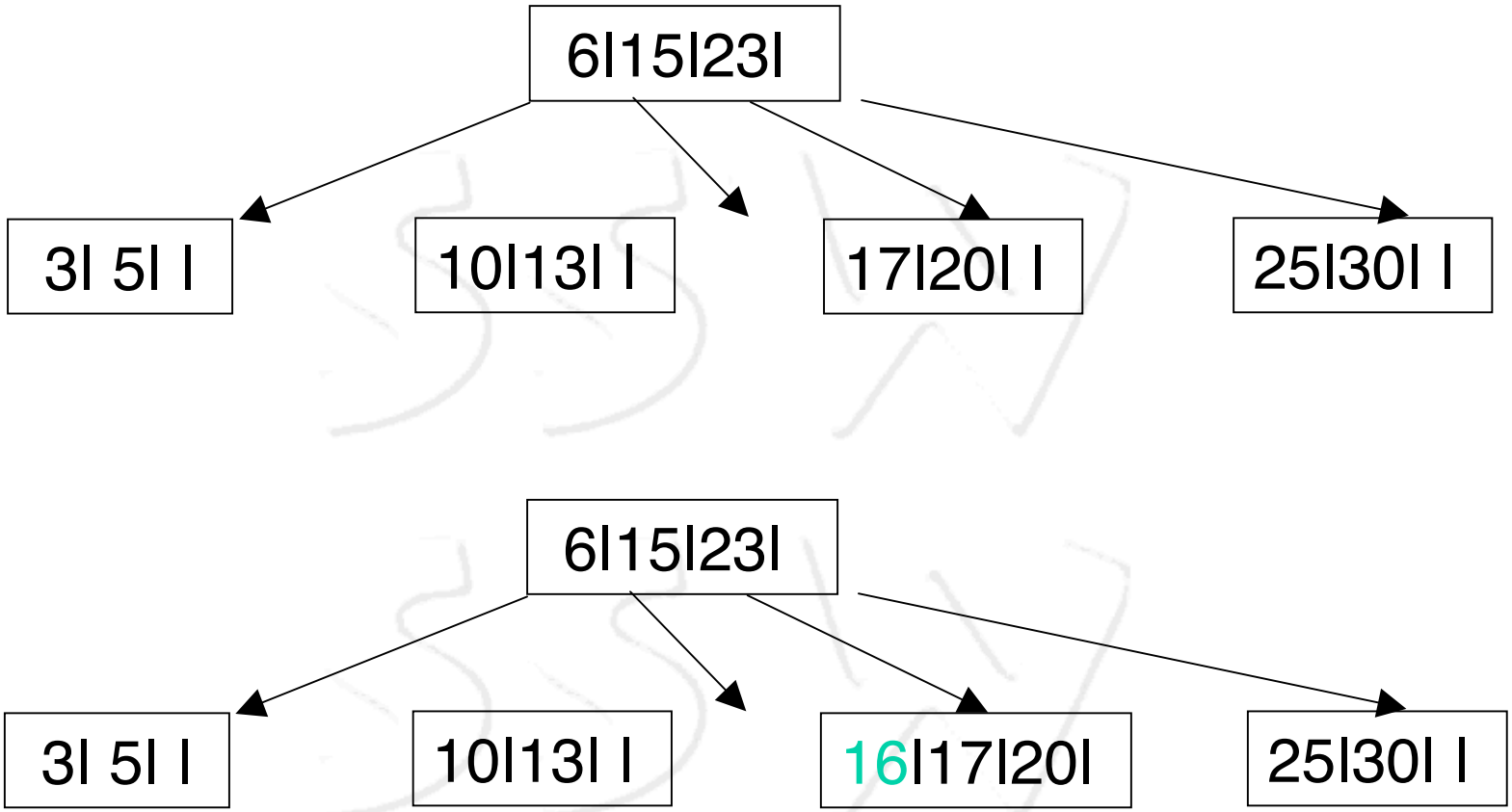
B-Baum

- Einfügen von: 16, 27, 40, 2, 4, 1
- Löschen von: 17, 10, 23



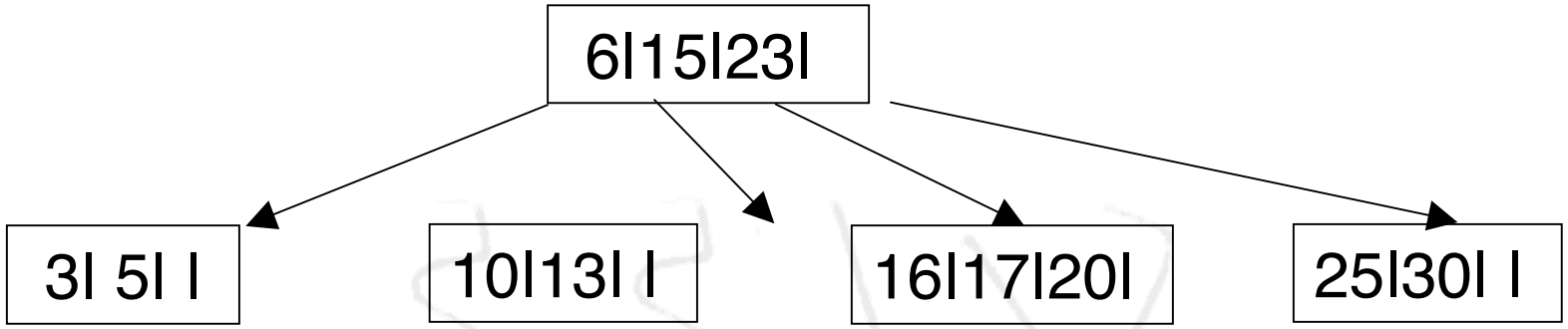
B-Baum

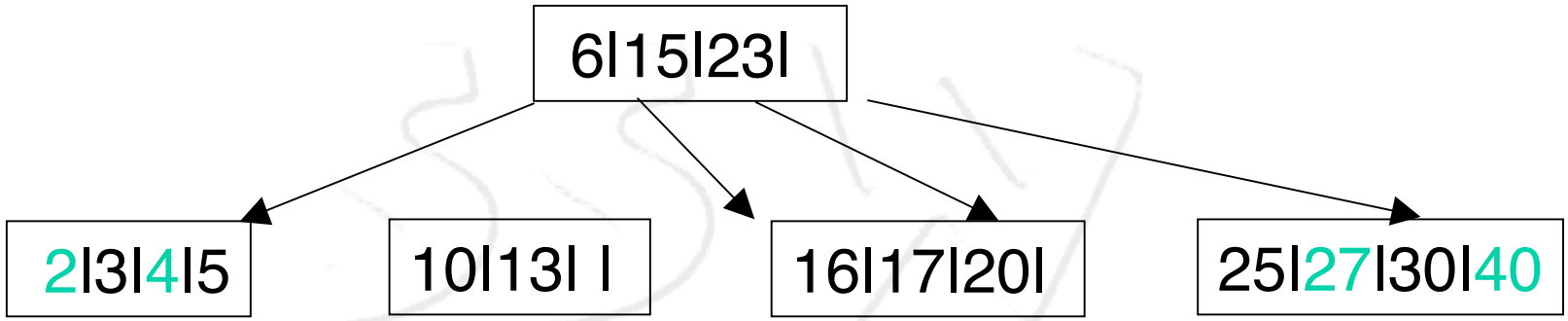
- Einfügen von: 16



B-Baum

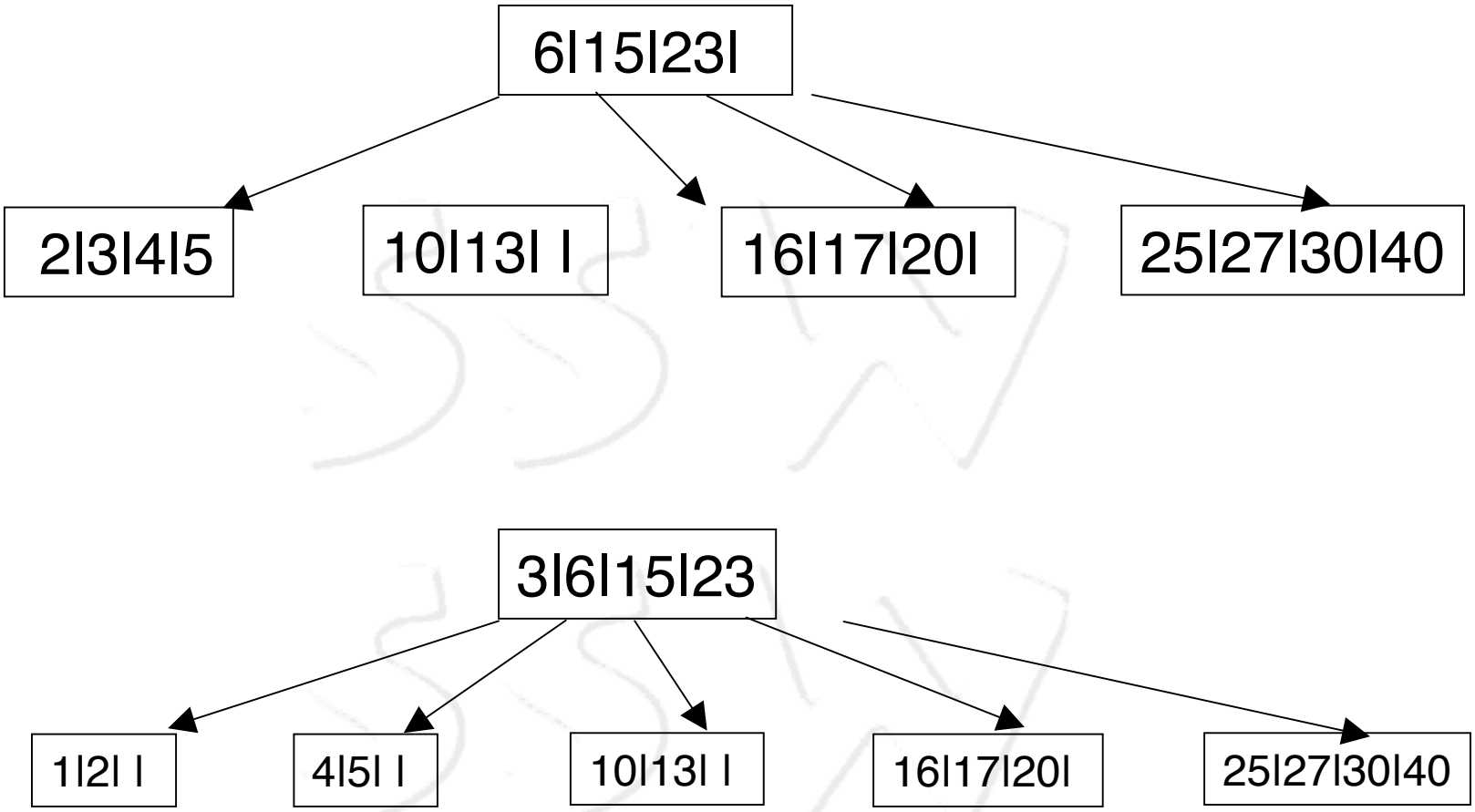
- Einfügen von: 27, 40, 2, 4





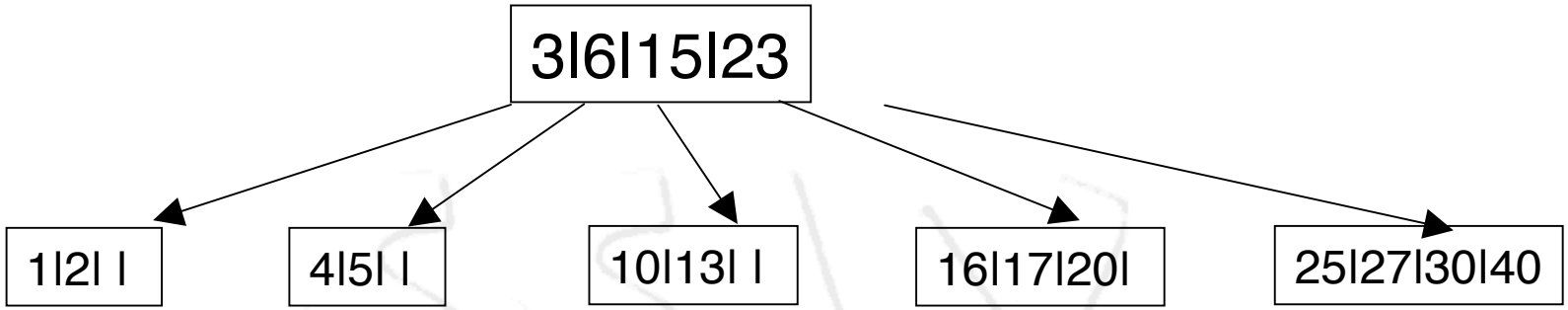
B-Baum

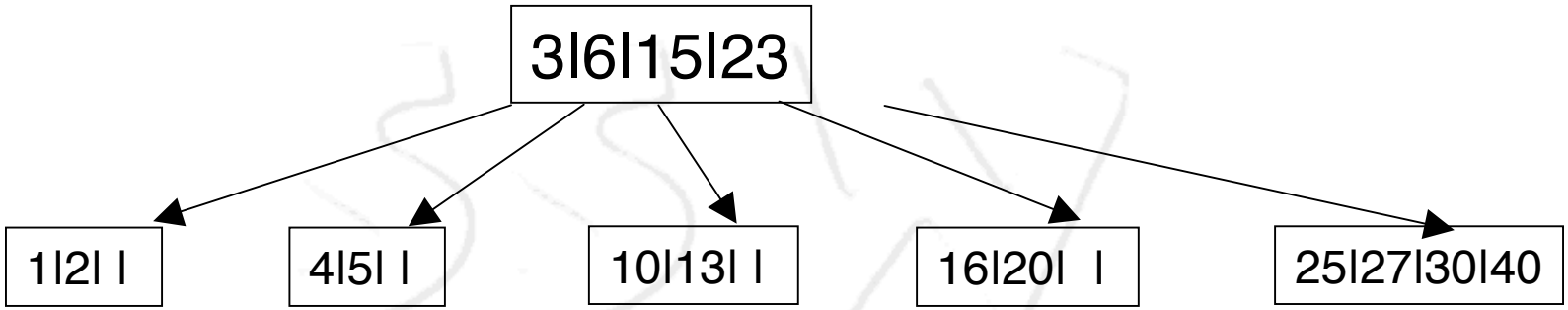
- Einfügen von: 1



B-Baum

- Löschen von: 17





B-Baum

- Löschen von: 10