VL06, Lösung 1

a) Die Höhe eines Baumes ist die maximale Länge aller Pfade von der Wurzel zu einem Blatt. Im gegebenen Baum beträgt die Höhe 5 (Pfad zu `201`).

b) Knoten, die keine Nachfolger haben, heißen Blatt. Der gegebene Baum hat somit 5 Blätter: `19`, `50`, `78`, `201` und `444`.

c) Ja, es handelt sich um einen Binärbaum, da jeder Knoten maximal 2 Nachfolger hat.

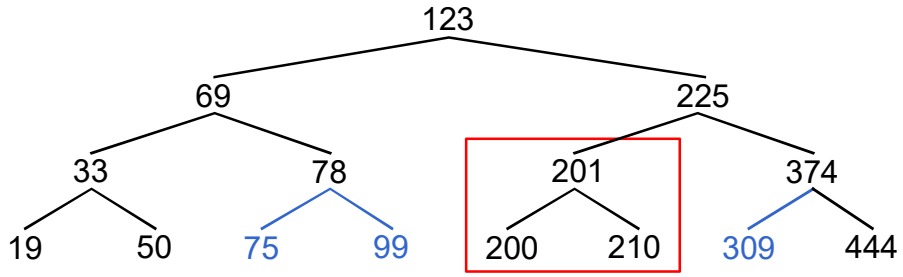
d) Ja, es handelt sich um einen Suchbaum, da für **jeden** Knoten die Suchbaumeigenschaft erfüllt ist: alle Schlüssel im jeweils linken Teilbaum sind kleiner und alle Schlüssel im jeweils rechten Teilbaum sind größer als der Schlüssel des aktuellen Knotens.

e) Algorithmus zur Schlüsselsuche (**nur für Suchbäume gültig**, siehe d):

1. Enthält der Teilbaum Knoten?
2. Wenn nein, ist der Schlüssel nicht im Baum enthalten. Ende!
3. Wenn ja, dann ist zu prüfen, ob der aktuelle Knoten (Wurzel des Teilbaums) den gesuchten Schlüssen enthält. Falls ja: Ende!
4. Ansonsten muss geprüft werden, ob der gesuchte Schlüssel kleiner oder größer ist als der Schlüssel im aktuellen Knoten
5. Ist er kleiner, muss rekursiv im linken Teilbaum weitergesucht werden
6. Ist er größer, muss rekursiv im rechten Teilbaum weitergesucht werden

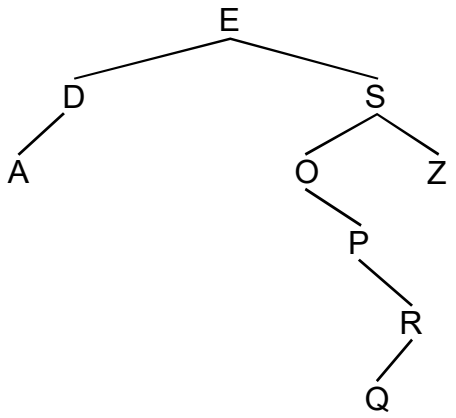
Suche nach `210`: `123-225-200-210` **GEFUNDEN**
Suche nach `50`: `123-69-33-50` **GEFUNDEN**
Suche nach `123`: `123` **GEFUNDEN**
Suche nach `371`: `123-225-364-444` **NICHT GEFUNDEN** (linker Teilbaum leer)

f) Der gegebene Baum ist nicht vollständig. Durch eine Rotation der Knoten `200-210-201` (rot) und durch Hinzufügen neuer Knoten (blau) kann der Baum vervollständigt werden, so dass er auf jedem Niveau die maximale Knotenzahl hat und sich alle Blätter auf derselben Ebene befinden:



VL06, Lösung 2

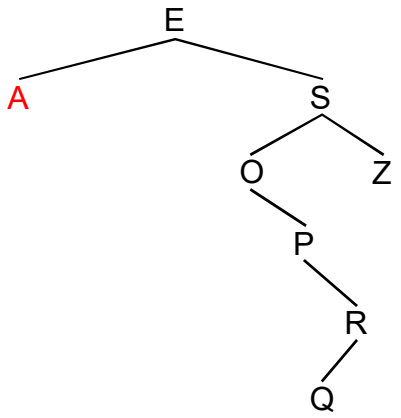
a) **Nach dem Einfügen aller Knoten:**



b) **Pre-Order:** E, D, A, S, O, P, R, Q, Z
In-Order: A, D, E, O, P, Q, R, S, Z
Post-Order: A, D, Q, R, P, O, Z, S, E

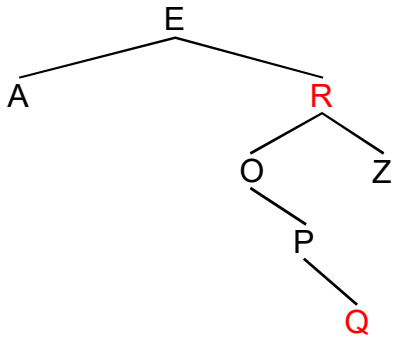
c) **Löschen des Knotens mit dem Schlüssel `D`:**

Der Knoten mit dem Schlüssel `D` hat nur einen Nachfolger. Daher muss lediglich der linke Nachfolger des Elternknotens (Knoten mit Schlüssel `E`) auf den Nachfolger des zu löschenden Knotens gesetzt werden.



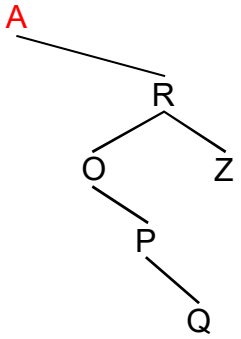
Löschen des Knotens mit dem Schlüssel `S`:

Der größte Schlüssel im linken Teilbaum des Knotens mit Wert `S` ist der Knoten mit Wert `R`. Der Schlüsselwert `S` wird durch `R` ersetzt. Der rechte Nachfolger des Knotens mit Wert `P` wird frei und muss jetzt auf den linken Nachfolger des ursprünglichen Knotens mit dem Schlüssel `R` verweisen. Da dieser Knoten der Knoten mit dem größten Schlüsselwert im linken Teilbaum war, konnte er keinen rechten Nachfolger haben.



Löschen des Knotens mit dem Schlüssel `E`:

Der Schlüsselwert des Wurzelknotens wird durch `A` ersetzt, da es sich hierbei um den größten (und einzigen) Schlüssel im linken Teilbaum handelt.



VL06, Lösung 3

Beachten Sie, dass die Abschnitte für Pre-Order, In-Order und Post-Order grundsätzlich gleich aufgebaut sind. Lediglich die Reihenfolge der drei Anweisungen „Linken Teilbaum durchsuchen“, „Rechten Teilbaum durchsuchen“ und „Aktuellen Knoten bearbeiten“ **sowie der Name der rekursiven Methode** sind jeweils unterschiedlich!

```
// Pre-Order
public String traversierePreOrder()
{
    return (wurzel != null) ?
        traversierePreOrder(wurzel) : "Der Baum ist leer.";
}

private String traversierePreOrder(final Knoten<T> einKnoten)
{
    assert(einKnoten != null);

    String result = "" + einKnoten.getDaten();

    if (einKnoten.getKnotenLinks() != null)
        result += traversierePreOrder(einKnoten.getKnotenLinks());

    if (einKnoten.getKnotenRechts() != null)
        result += traversierePreOrder(einKnoten.getKnotenRechts());

    return result;
}
```

```
    // In-Order
    public String traversiereInOrder()
    {
        return (wurzel != null) ?
            traversiereInOrder(wurzel) : "Der Baum ist leer.";
    }

    private String traversiereInOrder(final Knoten<T> einKnoten)
    {
        assert(einKnoten != null);

        String result = "";

        if (einKnoten.getKnotenLinks() != null)
            result += traversiereInOrder(einKnoten.getKnotenLinks());

        result += einKnoten.getDaten();

        if (einKnoten.getKnotenRechts() != null)
            result += traversiereInOrder(einKnoten.getKnotenRechts());

        return result;
    }

    // Post-Order
    public String traversierePostOrder()
    {
        return (wurzel != null) ?
            traversierePostOrder(wurzel) : "Der Baum ist leer.";
    }

    private String traversierePostOrder(final Knoten<T> einKnoten)
    {
        assert(einKnoten != null);

        String result = "";

        if (einKnoten.getKnotenLinks() != null)
            result += traversierePostOrder(einKnoten.getKnotenLinks());

        if (einKnoten.getKnotenRechts() != null)
            result += traversierePostOrder(einKnoten.getKnotenRechts());

        result += einKnoten.getDaten();

        return result;
    }
}
```

Für eine Ausgabe des Baums in umgekehrter Sortierreihenfolge müssen in der Methode `traversiereInOrder` die Aufrufe für den linken und rechten Teilbaum vertauscht werden.

```
if (einKnoten.getKnotenRechts() != null)
    result += traversiereInOrder(einKnoten.getKnotenRechts());

result += einKnoten.getDaten();

if (einKnoten.getKnotenLinks() != null)
    result += traversiereInOrder(einKnoten.getKnotenLinks());
```

Für `traversierePreOrder` lässt sich für den Baum aus Aufgabe 2a die rekursive Aufrufreihenfolge wie folgt darstellen:

```
|-Call: traversierePreOrder                Teilbaum mit Schlüssel E
|  |-Call: traversierePreOrder             Teilbaum mit Schlüssel D
|  |  |-Call: traversierePreOrder          Teilbaum mit Schlüssel A
|  |  |-return "A"
|  |-Exit: traversierePreOrder
|  |-return "DA"
|  |-Exit: traversierePreOrder
|  |-Call: traversierePreOrder             Teilbaum mit Schlüssel S
|  |  |-Call: traversierePreOrder          Teilbaum mit Schlüssel O
|  |  |  |-Call: traversierePreOrder       Teilbaum mit Schlüssel P
|  |  |  |  |-Call: traversierePreOrder    Teilbaum mit Schlüssel R
|  |  |  |  |  |-Call: traversierePreOrder Teilbaum mit Schlüssel Q
|  |  |  |  |  |-return "Q"
|  |  |  |  |-return "RQ"
|  |  |  |  |-Exit: traversierePreOrder
|  |  |  |-return "PRQ"
|  |  |  |-Exit: traversierePreOrder
|  |  |-return "OPRQ"
|  |  |-Exit: traversierePreOrder
|  |  |-Call: traversierePreOrder          Teilbaum mit Schlüssel Z
|  |  |-return "Z"
|  |  |-Exit: traversierePreOrder
|  |-return "SOPRQZ"
|  |-Exit: traversierePreOrder
|-return "EDASOPRQZ"
|-Exit: traversierePreOrder
```

VL06, Lösung 4

```
private int hoeheRek(final Knoten<T> einKnoten)
{
   int hoehe = 0;

   if (einKnoten != null)
      hoehe = Math.max(hoeheRek(einKnoten.getKnotenLinks()),
                       hoeheRek(einKnoten.getKnotenRechts())) + 1;

   return hoehe;
}
```

Java-Programme zu den Lösungen der Aufgaben 3 und 4 finden Sie zusätzlich in der Datei ML06-Aufgabe_3_4.zip.