Aufgabe 10

Sei B ein binärer Suchbaum. In jedem Knoten v von B wird ein Schlüssel $v.\text{key} \in \mathbb{N}$ gespeichert sowie Zeiger $v.\text{left}$, $v.\text{right}$ und $v.\text{parent}$ auf sein linkes Kind, auf sein rechtes Kind und auf seinen Elternknoten. Die Zeiger sind *nil*, wenn der entsprechende Nachbar nicht existiert. Für zwei Knoten u und v ist wie üblich der *Abstand* die Anzahl der Kanten auf dem kürzesten Pfad von u nach v.

Für einen Knoten w von B sei $B(w)$ der Teilbaum von B mit Wurzel w. Für zwei Knoten u und v von B ist w ein *gemeinsamer* Vorfahre, wenn u und v in $B(w)$ liegen. Wir suchen den niedrigsten gemeinsamen Vorfahren $ngV(u, v)$ von u und v, also einen gemeinsamen Vorfahren w, so dass für jeden Vorfahren w von u und v gilt, dass w in $B(w)$ liegt. Wir betrachten verschiedene Szenarien, in denen Sie jeweils den niedrigsten gemeinsamen Vorfahren von u und v berechnen sollen.

> **Exkurs: Lowest Common Ancestor**
> Als Lowest Common Ancestor (LCA) oder „letzter gemeinsamer Vorfahre“ wird in der Informatik und Graphentheorie ein Ermittlungskonzept bezeichnet, das einen gegebenen gewurzelten Baum von Datenstrukturen effizient vorverarbeitet, sodass anschließend Anfragen nach dem letzten gemeinsamen Vorfahren für beliebige Knotenpaare in konstanter Zeit beantwortet werden können. [a]
>
> [a] `https://de.wikipedia.org/wiki/Lowest_Common_Ancestor`

(a) Wir bekommen u und v als Zeiger auf die entsprechenden Knoten in B geliefert. Beschreiben Sie in Worten und in Pseudocode einen Algorithmus, der den niedrigsten gemeinsamen Vorfahren von u und v berechnet. Analysieren Sie die Laufzeit Ihres Algorithmus.

```
3  /**
4   * NBV = Niedrigster gemeinsamer Vorfahre.
5   *
6   * https://afteracademy.com/blog/lowest-common-ancestor-of-a-binary-tree
7   */
8  public class NGV {
9    static class Knoten {
10     int schlüssel;
11     Knoten links;
12     Knoten rechts;
13
14     public Knoten(int schlüssel) {
15       this.schlüssel = schlüssel;
16     }
17   }
18
19   /**
20    * ngV = niedrigster gemeinsamer Vorfahre
21    *
22    * @param wurzel  Der Wurzelkonten dies Binärbaums.
23    * @param knoten1 Der erste Knoten, dessen niedrigster gemeinsamer
      ↪  Vorfahre
24    *                gesucht werden soll.
25    * @param knoten2 Der zweite Knoten, dessen niedrigster gemeinsamer
      ↪  Vorfahre
26    *                gesucht werden soll.
```

```
27      *
28      * @return Der niedrigste gemeinsae Vorfahre der Knoten 1 und 2.
29      */
30     public static Knoten ngVRekursiv(Knoten wurzel, Knoten knoten1, Knoten
        ↪  knoten2) {
31       if (wurzel == null)
32         return null;
33       if (wurzel.equals(knoten1) || wurzel.equals(knoten2))
34         return wurzel;
35       Knoten links = ngVRekursiv(wurzel.links, knoten1, knoten2);
36       Knoten rechts = ngVRekursiv(wurzel.rechts, knoten1, knoten2);
37       if (links == null)
38         return rechts;
39       else if (rechts == null)
40         return links;
41       else
42         return wurzel;
43     }
44
45     /**
46      * <pre>
47      * {@code
48      *     20
49      *    / \
50      *   8   22
51      *  / \
52      * 4   12
53      *    / \
54      *   10  14
55      * }
56      * </pre>
57      *
58      * Beispiele von
59      * https://www.geeksforgeeks.org/lowest-common-ancestor-in-a-binary-
        ↪  search-tree/
60      *
61      * @param args Kommandozeilen-Argumente
62      */
63     public static void main(String[] args) {
64       Knoten wurzel = new Knoten(20);
65
66       Knoten knoten8 = new Knoten(8);
67       Knoten knoten22 = new Knoten(22);
68       Knoten knoten4 = new Knoten(4);
69       Knoten knoten12 = new Knoten(12);
70       Knoten knoten10 = new Knoten(10);
71       Knoten knoten14 = new Knoten(14);
72
73       wurzel.links = knoten8;
74       wurzel.rechts = knoten22;
75       wurzel.links.links = knoten4;
76       wurzel.links.rechts = knoten12;
77       wurzel.links.rechts.links = knoten10;
78       wurzel.links.rechts.rechts = knoten14;
79
80       System.out.println(ngVRekursiv(wurzel, knoten10, knoten14).schlüssel);
         ↪  // 12
81       System.out.println(ngVRekursiv(wurzel, knoten14, knoten8).schlüssel);
         ↪  // 8
82       System.out.println(ngVRekursiv(wurzel, knoten10, knoten22).schlüssel);
         ↪  // 20
83     }
```

```
84 }
```

(b) Wir bekommen u und v wieder als Zeiger auf die entsprechenden Knoten in B geliefert. Seien d_u, und d_v, die Abstände von u bzw. v zum niedrigsten gemeinsamen Vorfahren von u und v. Die Laufzeit Ihres Algorithmus soll $O(\max\{d_u, d_v\})$ sein. Dabei kann Ihr Algorithmus in jedem Knoten v eine Information v.info speichern. Skizzieren Sie Ihren Algorithmus in Worten.

(c) Wir bekommen die Schlüssel u.key und v.key. Die Laufzeit Ihres Algorithmus soll proportional zum Abstand der Wurzel von B zum niedrigsten gemeinsamen Vorfahren von u und v sein. Skizzieren Sie Ihren Algorithmus in Worten.