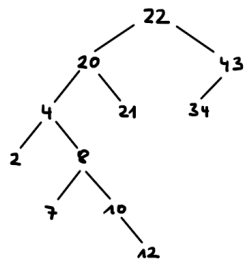
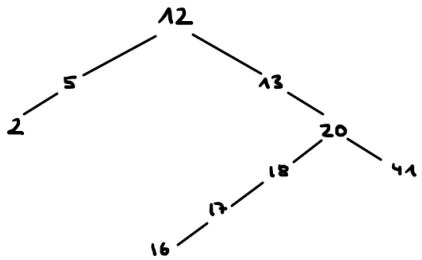
1. (3 Punkte) In einen leeren Suchbaum werden nacheinander die Zahlen 22, 43, 34, 20, 21, 4, 2, 8, 7, 10, 12 eingefügt. Zeichne den Suchbaum.

Lösung:



2. (3 Punkte) In einen leeren Suchbaum werden nacheinander die Zahlen 12, 13, 20, 5, 2, 41, 12, 18, 17, 16 eingefügt. Zeichne den Suchbaum.

Lösung:



Die zweite 12 wird nicht eingefügt, sie ist schon vorhanden.

3. (2 Punkte) In einen leeren Suchbaum werden nacheinander die Zahlen 50, 30, 70, 20, 40, 60, 80, 35 eingefügt. Gib jeweils an, mit welchen Zahlen die gesuchte Zahl bei `lookup` verglichen wird.

   a) Suche nach 35

   b) Suche nach 65

Lösung:

a) 50, 30, 40, 35. Die 35 wird gefunden.

b) 50, 70, 60. Rechts von der 60 ist kein Knoten mehr, die 65 ist nicht im Suchbaum.

4. (2 Punkte) Die Funktionen `insert`, `inorder` und `ebenen` sind wie im Unterricht definiert. Was wird ausgegeben?

```
baum = None
for x in [5, 3, 8, 1, 4, 9]:
    baum = insert(baum, x)
inorder(baum)
print()
print(ebenen(baum))
```

Lösung:

```
1 3 4 5 8 9
3
```

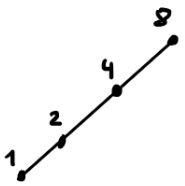
Der inorder-Durchlauf liefert die Zahlen sortiert. Der Baum hat die Ebenen 5 / 3, 8 / 1, 4, 9.

5. (2 Punkte) Gib eine Reihenfolge der Zahlen 1 bis 7 an, so dass beim Einfügen der Zahlen in dieser Reihenfolge in einen leeren Suchbaum ein vollkommen ausgeglichener Suchbaum entsteht.

Lösung: z.B. 4, 2, 1, 3, 6, 5, 7

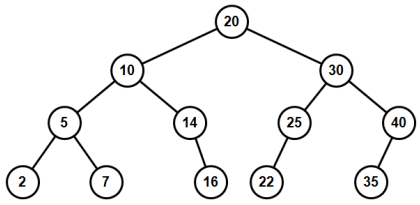
Die 4 muss zuerst kommen, die 2 vor 1 und 3, die 6 vor 5 und 7. Andere Reihenfolgen, die das erfüllen, sind ebenfalls richtig, z.B. 4, 6, 2, 7, 5, 3, 1.

6. (1 Punkt) Gib eine Reihenfolge der Zahlen an, so dass beim Einfügen in einen leeren Suchbaum der abgebildete Suchbaum entsteht.



Lösung: 8, 4, 2, 1 (die einzige mögliche Reihenfolge)

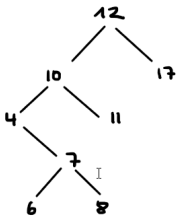
7. (2 Punkte) Gib eine Reihenfolge der Zahlen an, so dass beim Einfügen in einen leeren Suchbaum der abgebildete Suchbaum entsteht.



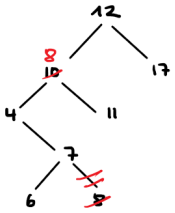
Lösung: z.B. 20, 10, 5, 2, 7, 14, 16, 30, 25, 22, 40, 35

Jede Zahl muss vor den Zahlen in ihren Teilbäumen kommen. Andere Reihenfolgen, die das erfüllen, sind ebenfalls richtig.

8. (2 Punkte) In dem folgenden Suchbaum wird die 10 gelöscht. Zeichne den Suchbaum nach dem Löschvorgang.

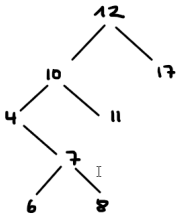


Lösung:

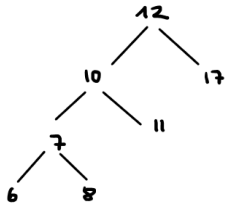


Die 10 hat zwei Kinder. Die größte Zahl im linken Teilbaum, die 8, rückt an ihre Stelle.

9. (2 Punkte) In dem folgenden Suchbaum wird die 4 gelöscht. Zeichne den Suchbaum nach dem Löschvorgang.

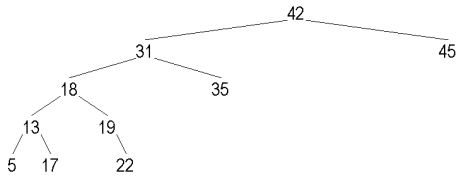


Lösung:

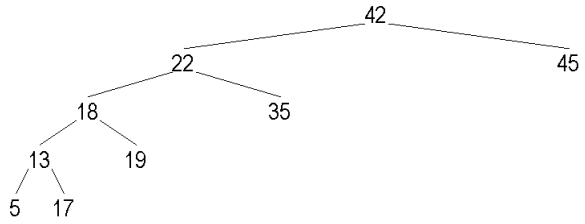


Die 4 hat nur ein Kind, die 7. Sie rückt mit ihrem Teilbaum an die Stelle der 4.

10. (2 Punkte) In dem folgenden Suchbaum wird die 31 gelöscht. Zeichne den Suchbaum nach dem Löschvorgang.



Lösung:



Die 31 hat zwei Kinder. Die größte Zahl im linken Teilbaum, die 22, rückt an ihre Stelle.

11. (3 Punkte) Schreibe eine Funktion `kleinstes(node)`, die die kleinste Zahl in einem nicht leeren Suchbaum zurückgibt. `node` ist die Wurzel des Suchbaums, die Knoten haben die Attribute `val`, `left` und `right`. Nutze aus, dass es sich um einen Suchbaum handelt.

Lösung:

```
def kleinstes(node):
    while node.left is not None:
        node = node.left
    return node.val
```

Die kleinste Zahl findet man, indem man immer nach links geht, bis es nicht mehr weitergeht. Eine rekursive Lösung ist ebenfalls richtig.

12. (3 Punkte) In einem Suchbaum sind 1023 verschiedene Zahlen gespeichert.

    a) Gib an, mit wie vielen Zahlen `lookup` höchstens vergleichen muss, wenn der Suchbaum vollkommen ausgeglichen ist.

    b) Gib an, mit wie vielen Zahlen `lookup` höchstens vergleichen muss, wenn der Suchbaum so ungünstig wie möglich ist. Beschreibe, wie dieser Fall entstehen kann.

Lösung:

a) Ein vollkommen ausgeglichener Suchbaum mit 10 Ebenen hat $2^{10} - 1 = 1023$ Knoten. Es sind also höchstens 10 Vergleiche nötig.

b) 1023 Vergleiche. Das passiert, wenn die Zahlen sortiert eingefügt werden. Dann entsteht eine Kette mit 1023 Ebenen.