# Blatt 08

## Aufgabe 1: Kollisionsbehandlung

Gegeben sei die folgende Hashfunktion: $f1(k) = (3k + 3)\%11$

$$\boxed{13, 18, 44, 23, 2, 47, 22, 11}$$

Lineare Sondierung (linear probing)

| Index | Value | Value (lineare Sondierung) |
|---|---|---|
| 0 | | |
| 1 | **47**, f(47)=(141 + 3)%11=1 | **47**, f(47)=(141 + 3)%11=1 |
| 2 | **18**, f(18)=(54+3)%11=2 | **18**, f(18)=(54+3)%11=2 |
| 3 | **44**, f(44)=(132+3)%11=3 ; **22**, f(22)=(66+3)%11=3 ; **11**, f(11)=(33+3)%11=3 | **44**, f(44)=(132+3)%11=3 ; |
| 4 | | **22**, f(22)=(66+3)%11=3 ; |
| 5 | | **11**, f(11)=(33+3)%11=3 |
| 6 | **23,** f(23)=(69+3)%11=6 | **23**, f(23)=(69+3)%11=6 |
| 7 | | |
| 8 | | |
| 9 | **13,** f(13)=(39+3)%11=9 ; **2,** f(6+3)%11=9 | **13**, f(13)=(39+3)%11=9 ; |
| 10 | | **2**, f(6+3)%11=9 |

Externe Verkettung (seperate chaining)

| Index | Value (externe Verkettung) | Kette | Kette |
|---|---|---|---|
| 0 | | | |
| 1 | **47,** f(47)=(141 + 3)%11=1 | | |

| Index | Value (externe Verkettung) | Kette | Kette |
|---|---|---|---|
| 2 | **18**, f(18)= (54+3)%11=2 | | |
| 3 | **44**, f(44)= (132+3)%11=3 ; | **22**, f(22)= (66+3)%11=3 ; | **11**, f(11)=(33+3)%11=3 |
| 4 | | | |
| 5 | | | |
| 6 | **23**, f(23)= (69+3)%11=6 | | |
| 7 | | | |
| 8 | | | |
| 9 | **13**, f(13)= (39+3)%11=9 ; | **2**, f(6+3)%11=9 | |
| 10 | | | |

Doppeltes Hashing (double hashing):

$f1(k) = (3k + 3)\%11$

$f2(k) = k \mod 10$ <= Zweite Hash Funktion

$h_j(k) = (f1(k) + j * f2(k)) \mod 11$, $j$ ist dabei Anzahl der bereits ausprobierten Indexes.

| Index | $f1(k)$ | $f2(k)$ | $h_j(k) = (f1(k) + j * f2(k)) \mod 11$ |
|---|---|---|---|
| 0 | | | **2,** h_j(2) = (9 + 1*2) mod 11 = 0 |
| 1 | **47**, f(47)=(141 + 3)%11=1 | **11**, f2(11)= 11 mod 10 = 1 | **47** |
| 2 | **18**, f(18)= (54+3)%11=2 | **2**, f2(2) = 2 mod 10 = 2; **22**, f2(22) = 22 mod 10 = 2 | **18** |
| 3 | **44**, f(44)= (132+3)%11=3 ; **22**, f(22)=(66+3)%11=3 ; **11**, f(11)= (33+3)%11=3 | **13**, f2(13) = 13 mod 10 = 3; **23**, f2(23) = 23 mod 10 = 3 | **44** |

| Index | $f1(k)$ | $f2(k)$ | $h_j(k) = (f1(k) + j * f2(k)) \mod 11$ |
|---|---|---|---|
| 4 | | **44**, f2(44) = 44 mod 10 = 4 | **11,** h_j(11) = (3 + 1*1) mod 11 = 4 |
| 5 | | | **22,** h_j(22) = (3 + 1*2) mod 11 = 3 |
| 6 | **23**, f(23)= (69+3)%11=6 | | **23** |
| 7 | | **47**, f2(47) = 47 mod 10 =7 | |
| 8 | | **18**, f2(18) = 18 mod 10=8 | |
| 9 | **13**, f(13)= (39+3)%11=9 ; **2**, f(6+3)%11=9 | | **13** |
| 10 | | | |

## Aufgabe 2

```
public static Set<Integer> intersection(final Set<Integer> set1, final Set<Integer> se
t2) {
        Set<Integer> set = new HashSet<>();
        for (Integer i:set2) {
            if (set1.contains(i)){
                set.add(i);
            }
        }
        return set;
    }


    // do not change the input sets
public static Set<Integer> union(final Set<Integer> set1, final Set<Integer> set2) {
        Set<Integer> set = new HashSet<>();
        set.addAll(set1);
        for (Integer i:set2) {
            if (!set.contains(i)){
                set.add(i);
            }
        }
        return set;
    }

    // do not change the input sets
public static Set<Integer> difference(final Set<Integer> set1, final Set<Integer> set
2) {
        Set<Integer> set = new HashSet<>();
```

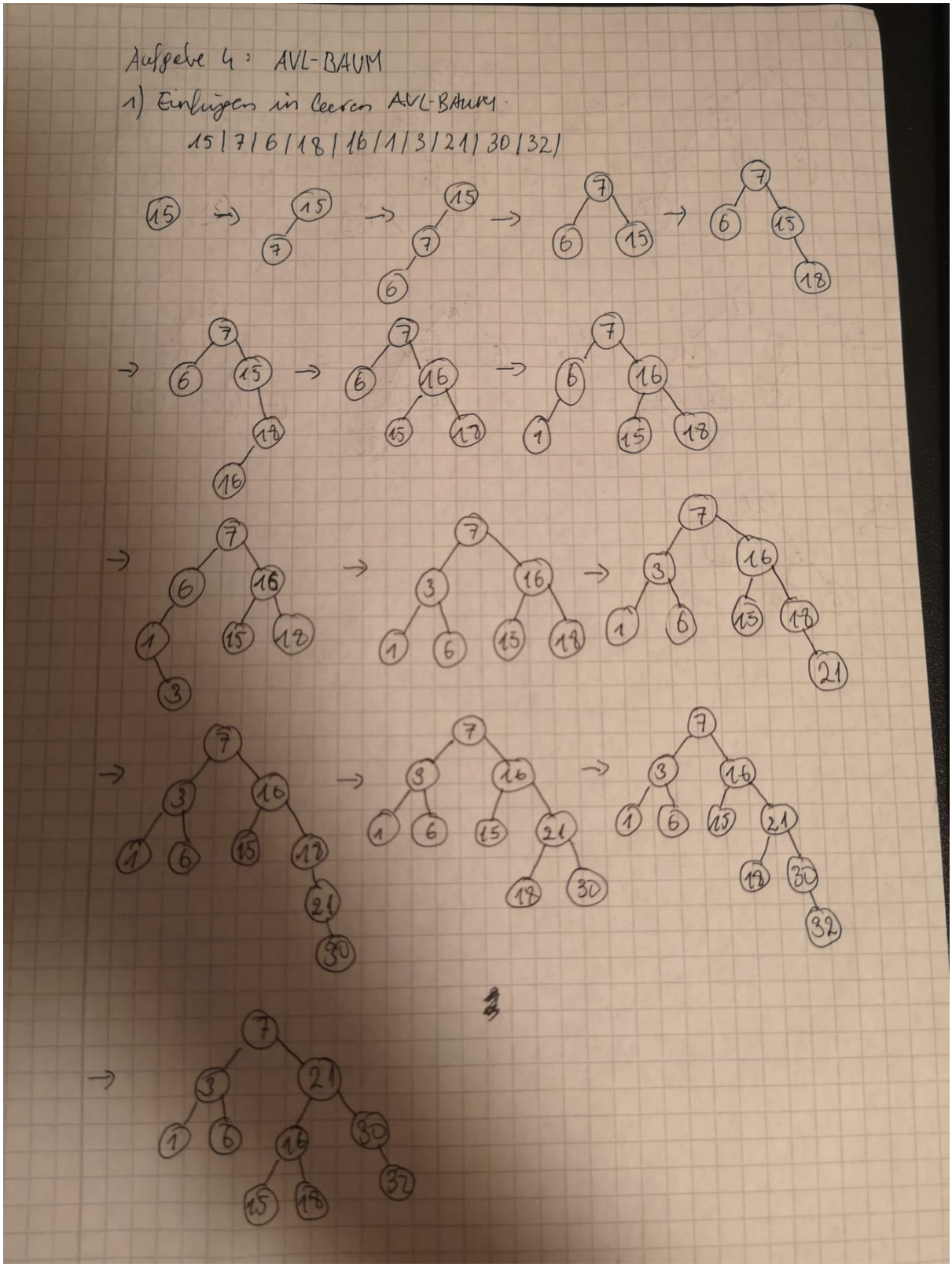
```
        for (Integer i:set1) {
            if (set1.contains(i)){
                set.add(i);
            }
        }
        return set;
    }

    // do not change the input sets
public static Set<Integer> symmetricDifference(final Set<Integer> set1, final Set<Inte
ger> set2) {
        Set<Integer> set = new HashSet<>();
        for (Integer i:set2) {
            if (!set1.contains(i)){
                set.add(i);
            }
        }
        for (Integer i:set1) {
            if (!set2.contains(i)){
                set.add(i);
            }
        }
        return set;
    }
```

## Aufgabe 3

1. HashMap und Arraylist. Die Map ist alphabetisch sortiert (Key: A-Z) und zu jedem Buchstabe eine Array List mit den Bücher des Anfangsbuchstaben. So kann Beim ersten eintippen die Suche verkleinert werden und nun muss man die ArrayList filtern mit den eingegebenen buchstabenfolge.
2. Ein sortiertes Array ist von Vorteil.
3. Um festzustellen, ob $n \leq k$ ist, dann braucht es eine Laufzeitkomplexität von $n * k + N$ um dies zu ermitteln. Es muss zuerst n herausgefunden werden um dann mit $k$ zu vergleichen. Dabei muss $n$ die Liste einmal durchlaufen.
4. $\mathcal{O}(n)$, da man $\mathcal{O}(1)$ für den ersten Buchstaben braucht und dann $\mathcal{O}(n)$ um die neue Liste der zutreffenden Bücher zu erstellen. (contains)

## Aufgabe 4

1. Einfügen in leeren AVL Baum:




2. Was ist die Höhe des resultierenden Baumes?
    a. 4
3. Führen Sie eine inorder Traversierung durch was fällt ihnen auf?

a. $\boxed{1|3|6|7|15|16|18|21|30|32}$

Die Elemente werden in geordneter Reihenfolge zurückgegeben.

4. Lösche die folgenden Elemente aus dem AVL-Baum:

$\boxed{18|15|1|3|30}$