Day 8: Resonant Collinearity

Die heutige Schwierigkeit hängt stark vom individuellen Wissenstand von Java ab. Für eine effiziente Lösung ist es einerseits wichtig, sich vom zweidimensionalen Karten-Modell zu lösen und alles nur auf "Zahlenebene" zu lösen. Zum anderen sollte man am besten eine **HashMap** und ein **Set** (Menge) als Datenstrukturen nutzen. Durch diese Vielzahl an komplexeren Datenstrukturen würde ich die durchschnittliche Schwierigkeit dieses Tages recht hoch einschätzen.

Teil 1

Es sei hier nur die (kommentierte) Lösung gezeigt:

Lösungsvorschlag

```
public void partOne() {
    int width = inputLines.get(0).length();
    int height = inputLines.size();

    // Eine HashMap, die pro Character alle zugehörigen Antennen-Koordinaten
(je als zweistelliges Array) speichert
    HashMap<Character, ArrayList<int[]>> antennas = new HashMap();

    // speichere direkt jede Antenne in der HashMap
    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < width; x++) {
            char c = line.charAt(x);
            if (c != '.') {
                if (!antennas.containsKey(c)) {
                    antennas.put(c, new ArrayList());
                }
                antennas.get(c).add(new int[]{x,y});
            }
        }
    }

    //Speichere die Antinodes-Koordinaten in einem Set (erlaubt keine
Dopplungen)
    Set<List<Integer>> antinodes = new HashSet();

    // Greife dir der Reihe nach je alle Antennen, die zusammengehören
(derselbe Name)
    for (ArrayList<int[]> coords: antennas.values()) {
        // Iteriere in einer doppelten Schleife über alle paarweisen
Kombiationen aller Antennen desselben Namens
        for (int i = 0; i < coords.size(); i++) {
            for (int j = i+1; j < coords.size(); j++) {
```

```
                int[] antA = coords.get(i);
                int[] antB = coords.get(j);

                // Berechne die Distanz der zwei Antennen
                int diffX = antA[0] - antB[0];
                int diffY = antA[1] - antB[1];

                // Berechne, ob die erste benachbare Koordinate erlaubt ist
                int newX1 = antA[0]+diffX;
                int newY1 = antA[1]+diffY;
                if (newX1 >= 0 && newX1 < width && newY1 >= 0 && newY1 <
height) {
                    antinodes.add(Arrays.asList(newX1, newY1));
                }

                // Berechne, ob die zweite benachbare Koordinate erlaubt ist
                int newX2 = antB[0]-diffX;
                int newY2 = antB[1]-diffY;
                if (newX2 >= 0 && newX2 < width && newY2 >= 0 && newY2 <
height) {
                    antinodes.add(Arrays.asList(newX2, newY2));
                }
            }
        }
    }

    System.out.println(antinodes.size());
}
```

Teil 2

Für Teil 2 sind nur wenige Änderungen nötig. In der Lösung befinden sich Kommentare nur dort, wo eine Änderung/Erweiterung nötig ist.

Lösungsvorschlag

```
public void partTwo() {
    int width = inputLines.get(0).length();
    int height = inputLines.size();

    HashMap<Character, ArrayList<int[]>> antennas = new HashMap();

    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < width; x++) {
            char c = line.charAt(x);
            if (c != '.') {
```

```
                if (!antennas.containsKey(c)) {
                    antennas.put(c, new ArrayList());
                }
                antennas.get(c).add(new int[]{x,y});
            }
        }
    }

    Set<List<Integer>> antinodes = new HashSet();

    for (ArrayList<int[]> coords: antennas.values()) {
        for (int i = 0; i < coords.size(); i++) {
            for (int j = i+1; j < coords.size(); j++) {
                int[] antA = coords.get(i);
                int[] antB = coords.get(j);
                // nun zählen auch die Antennenkoordinaten selbst als
Antinodes
                antinodes.add(Arrays.asList(antA[0], antA[1]));
                antinodes.add(Arrays.asList(antB[0], antB[1]));

                int diffX = antA[0] - antB[0];
                int diffY = antA[1] - antB[1];

                // von antA positiv laufen
                // laufe solange, bis man sich außerhalb der Map befindet
                int mult = 1;
                while(true) {
                    int newX = antA[0]+(diffX*mult);
                    int newY = antA[1]+(diffY*mult);
                    if (newX >= 0 && newX < width && newY >= 0 && newY <
height) {
                        antinodes.add(Arrays.asList(newX, newY));
                    } else {
                        break;
                    }
                    mult++;
                }

                // von antB negativ laufen
                mult = 1;
                while(true) {
                    int newX = antB[0]-(diffX*mult);
                    int newY = antB[1]-(diffY*mult);
                    if (newX >= 0 && newX < width && newY >= 0 && newY <
height) {
                        antinodes.add(Arrays.asList(newX, newY));
                    } else {
                        break;
                    }
                    mult++;
                }
```

```
                }
            }
        }

        System.out.println(antinodes.size());
}
```