# Day 10: Hoof It

Dieser Tag ist wieder ziemlich einfach und lässt sich mit **Rekursion** straight-forward lösen.

## Teil 1

Grundsätzliche Vorgehensweise: Starte pro 0-Höhe einen rekursiven Aufruf, der jeweils wiederum in alle 4 möglichen Richtungen einen weiteren rekursiven Aufruf startet.

Tipps zur Vorgehensweise:

- Speichere den Input in einem zweidimensionalen int-Array. Jeden `char` c kannst du mit `(int)(c-'0')` in einen `int` casten.
- Iteriere anschließend über jede Koordinate des Arrays. Wenn eine Koordinate 0 ist (= Startpunkt eines Trails), dann starte dort einen rekursiven Aufruf. Der rekursive Aufruf benötigt folgende Parameter:
  1. Bei diesem **ersten** Aufruf eine **Kopie** der Karte (des int-Arrays). Hintergrund dazu: Wenn der rekursive Aufruf ein Ende eines Trails (die Zahl 9) gefunden hat, so muss er diese 9 "markieren", damit dieses Trail-Ende bei einem benachbarten rekursiven Aufruf nicht erneut gefunden/gezählt wird. Diese Markierung darf aber nur pro Startpunkt (pro ausgehender 0) gelten - ein Trail der von einer neuen Koordinate (einer neuen 0) beginnt, der darf wieder zum selben Ziel (9) führen.
  2. Die zu erwartende Höhe an der nächsten Koodinate (siehe nächste Parameter).
  3. Nächste x-Koordinate
  4. Nächste y-Koordinate
- Pro rekursivem Aufruf gilt nun folgendes:
  - Prüfe auf den Basis-Fall, ob die übergebenen Koordinaten außerhalb der Map liegen. Falls ja, dann gib eine 0 zurück, da mit diesem rekursiven Aufruf ganz offensichtlich kein weiterer Weg zum Ziel gefunden werden kann.
  - Prüfe auf den Basisfall, dass die Höhe der übergebenen Koordinate nicht der ebenfalls übergebenen erwarteten Höhe entspricht → ebenfalls 0 zurückgeben. *Für alle nun nachfolgenden Bedingungen weiß man dann entsprechend, dass die erwartete Höhe mit der vorgefundenen Höhe übereinstimmt.*
  - Prüfe auf den Basisfall, dass die Höhe der Zahl 9 entspricht. Gib 1 zurück, da **ein** weiter Trail gefunden wurde.
  - Ansonsten sind wir auf dem richtigen Weg, aber noch nicht am Ziel und müssen entsprechend in alle 4 möglichen Richtungen weiterlaufen. Ruf also der Reihe nach die Rekursion 4x auf, wobei du jedes Mal eine der Koordinaten um +/- 1 veränderst und die nächste erwartete Zahl/Höhe um eins erhöhst. Ermittle die Summe der Rückgabewerte aller 4 rekursiven Aufrufe. So viele neue Trails konnten ermittelt werden und diese Zahl muss entsprechend zurückgegeben werden.

Lösungsvorschlag:

```
public void partOne() {
    // Instanzvariablen (!) für Höhe und Breite der Karte
```

```
    width = inputLines.get(0).length();
    height = inputLines.size();

    // Karte als int-Array
    int[][] map = new int[width][height];

    // Speichere den Input als int-Array
    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < width; x++) {
            map[x][y] = (int)(line.charAt(x)-'0');
        }
    }

    int sumOfTrails = 0; // speichert die Summe aller Trails

    // Iteriere über jede Koordinate
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            // Wenn die Koordinate mit 0 beginnt, dann starte den rekursiven
Aufruf
            if (map[x][y] == 0) {
                /* Wichtig: Starte den Aufruf mit einer KOPIE der Map, da
die gefundenen
                * Trail-Enden pro Aufruf markiert werden müssen. Diese
Markierungen dürfen
                * im nächsten Aufruf aber nicht mehr vorhanden sein.
                */
                sumOfTrails += trail(copyMap(map), 0, x, y);
            }
        }
    }

    System.out.println(sumOfTrails);
}

/**
 * Läuft rekursiv jeden möglichen Weg von bestimmten Startpunkt 0 zu jeder
möglichen 9.
 *
 * @param map Die Karte
 * @param h Erwartete Höhe des aktuellen Iterationsschritts
 * @param x aktuelle x-Koordinate
 * @param y aktuelle y-Koordinate
 * @return Die Anzahl der gefunden Trail-Enden.
 */
private int trail(int[][] map, int h, int x, int y) {
    // Basisfall: Aktuelle Koordinate (x,y) liegt außerhalb der Map
    if (x < 0 || x >= width || y < 0 || y >= height) {
        return 0;
```

```
    }

    // Basisfall: Aktuelle Koordinate (x,y) hat nicht die erwartete
Höhe/Wert
    if (map[x][y] != h) {
        return 0;
    }

    // (Erfolgreicher) Basisfall: Ein Ende des Trails wurde erreicht!
    if (map[x][y] == 9) {
        map[x][y] = -1; // Markiere das Ende, damit dieses Trail-Ende nicht
nochmals gefunden wird!
        return 1; // return 1, damit dieses gefundene Trail-Ende gezählt
wird.
    }

    // Zähle über alle 4 Richtungen von der aktuellen Koordinate die Summe
aller gefundenen Trail-Enden
    int sumOfTrails = 0;
    sumOfTrails += trail(map, h+1, x+1, y);
    sumOfTrails += trail(map, h+1, x-1, y);
    sumOfTrails += trail(map, h+1, x, y+1);
    sumOfTrails += trail(map, h+1, x, y-1);
    return sumOfTrails;
}

/**
 * Erstellt eine Kopie der Karte
 */
private int[][] copyMap(int[][] map) {
    // Erstelle int-Array mit denselben Maßen
    int[][] copy = new int[width][height];

    // Kopiere jede Koordinate
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            copy[x][y] = map[x][y];
        }
    }
    return copy;
}
```

## Teil 2

Für den Teil 2 muss nur eine **winzige Kleinigkeit** angepasst werden.

In Teil 1 geht es darum, pro Startpunkt zu zählen, wie viele Trail-Enden man erreichen kann. Jedes Trail-Ende darf also nur einmal erreicht werden. Daher mussten wir da auch markieren, falls wir ein Ende erreicht hatten.

In Teil 2 geht es darum, pro Startpunkt zu zählen, wie **oft** ein Trail-Ende erreicht werden kann. Jedes Trail-Ende darf also von "benachbarten" rekursiven Aufrufen beliebig oft erreicht werden.

- Es muss daher nun **kein** Trail-Ende mehr als "bereits gefunden" markiert werden.
- Da nichts mehr markiert bzw. am Array "manipuliert" werden muss, musst du den ersten rekursiven Aufruf auch nicht mehr mit einer Kopie des Arrays starten.

Achte bei den Aufrufen der rekursiven Methoden darauf, dass du überall die korrekten, neuen Methoden für Teil 2 aufrufst.

Lösungsvorschlag

```
public void partTwo() {
    // Instanzvariablen (!) für Höhe und Breite der Karte
    width = inputLines.get(0).length();
    height = inputLines.size();

    // Karte als int-Array
    int[][] map = new int[width][height];

    // Speichere den Input als int-Array
    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < width; x++) {
            map[x][y] = (int)(line.charAt(x)-'0');
        }
    }

    int sumOfTrails = 0; // Zählt die Summe aller gefundenen Trails

    // Finde jeden Startpunkt eines Trails
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            if (map[x][y] == 0) {
                sumOfTrails += trail2(map, 0, x, y); // Starte den
rekursiven Aufruf - diesmal OHNE Kopie
            }
        }
    }
    System.out.println(sumOfTrails);
}

private int trail2(int[][] map, int h, int x, int y) {
    // Basisfall: Aktuelle Koordinate (x,y) liegt außerhalb der Map
    if (x < 0 || x >= width || y < 0 || y >= height) {
        return 0;
    }

    // Basisfall: Aktuelle Koordinate (x,y) hat nicht die erwartete
```

```
Höhe/Wert
    if (map[x][y] != h) {
        return 0;
    }

    // (Erfolgreicher) Basisfall: Ein Ende des Trails wurde erreicht!
    if (map[x][y] == 9) {
        return 1; // return 1, damit dieses gefundene Trail-Ende gezählt
wird.
    }

    // Zähle über alle 4 Richtungen von der aktuellen Koordinate die Summe
aller gefundenen Trail-Enden
    int sumOfTrails = 0;
    sumOfTrails += trail2(map, h+1, x+1, y);
    sumOfTrails += trail2(map, h+1, x-1, y);
    sumOfTrails += trail2(map, h+1, x, y+1);
    sumOfTrails += trail2(map, h+1, x, y-1);
    return sumOfTrails;
}
```