

Tag 18 - Lavaduct Lagoon

- [Variante 1](#)

Ich habe für Teil 1 und Teil 2 verschiedene Lösungswege genutzt.

Lösungshinweise Teil 1

- Finde zunächst heraus, wie groß der auszugrabende Bereich in die Breite und die Höhe sein muss.
- Erstelle ein `char[][]`-Array mit der passenden Größe.
- Trage den Input-Befehlen folgend ein 't' für "Trench" (Graben) in das `char[][]`-Array an jeder nötigen Stelle ein. Dadurch erhältst du den kompletten geschlossenen Verlauf des äußeren Grabens.
- Abschließend füllst du von jedem äußeren Pixel an den Kanten des `char[][]`-Arrays per Breitensuche die leeren Felder mit einem 'o' für "outside". Danach ist jedes leere Feld ein inneres Feld. Die Summe aller Lava-Pool-Felder ist die summe aller leeren Felder und aller 't'-Felder.

Lösungsvorschlag

```
public int partOne() {
    trenchList = new ArrayList();

    int offsetX = 0;
    int offsetY = 0;
    int largestX = 0;
    int largestY = 0;

    int lastX = 0;
    int lastY = 0;
    trenchList.add(new int[]{0,0});

    // speichere den offset für x und y
    for (String line: inputLines) {
        char dir = line.split(" ")[0].charAt(0);
        int steps = Integer.parseInt(line.split(" ")[1]);
        String color = line.split("[()") [1].split("[)]") [0];

        while (steps > 0) {
            if (dir == 'R') {
                trenchList.add(new int[]{lastX + 1, lastY});
                lastX++;
            } else if (dir == 'L') {
                trenchList.add(new int[]{lastX - 1, lastY});
                lastX--;
            } else if (dir == 'U') {
                trenchList.add(new int[]{lastX, lastY - 1});
                lastY--;
            }
        }
    }
}
```

```
    } else {
        trenchList.add(new int[]{lastX, lastY + 1});
        lastY++;
    }
    steps--;

    if (lastX < offsetX) {
        offsetX = lastX;
    }
    if (lastY < offsetY) {
        offsetY = lastY;
    }
    if (lastX > largestX) {
        largestX = lastX;
    }
    if (lastY > largestY) {
        largestY = lastY;
    }
}
}

breite = largestX - offsetX + 1;
hoehe = largestY - offsetY + 1;
map = new char[breite][hoehe];

// trage für die trench ein 't' in die map ein
for (int[] k: trenchList) {
    map[k[0]-offsetX][k[1]-offsetY] = 't';
}

for (int x = 0; x < breite; x++) {
    if (map[x][0] == '\u0000') {
        fillQueue.push(new int[]{x, 0});
        fillOutside();
    }
    if (map[x][hoehe-1] == '\u0000') {
        fillQueue.push(new int[]{x, hoehe-1});
        fillOutside();
    }
}
for (int y = 0; y < hoehe; y++) {
    if (map[0][y] == '\u0000') {
        fillQueue.push(new int[]{0, y});
        fillOutside();
    }
    if (map[breite-1][y] == '\u0000') {
        fillQueue.push(new int[]{breite-1, y});
        fillOutside();
    }
}
}
```

```
int insideCounter = 0;
for (int x = 0; x < breite; x++) {
    for (int y = 0; y < hoehe; y++) {
        if (map[x][y] != 'o') {
            insideCounter++;
        }
    }
}

return insideCounter;
}

private void fillOutside() {
    while (!fillQueue.isEmpty()) {
        int[] k = fillQueue.pop();
        int x = k[0];
        int y = k[1];

        if (x < 0 || y < 0 || x == breite || y == hoehe || map[x][y] !=
'\u0000') {
            continue;
        } else {
            map[x][y] = 'o';
            fillQueue.push(new int[]{x + 1, y});
            fillQueue.push(new int[]{x - 1, y});
            fillQueue.push(new int[]{x, y + 1});
            fillQueue.push(new int[]{x, y - 1});
        }
    }
}
```

Lösungshinweise Teil 2

- Für Teil 2 werden die Zahlen zu groß, sodass die Lösung aus Teil 1 nicht mehr ausreicht. Die Hauptvorgehensweise ist folgende:
- ✖
- Gehe der Reihe nach die Befehle entlang. Trage aber keine Kreuze in eine 2-dimensionale Matrix, sondern betrachte nur die einzelnen Spalten. Wenn

From:
<https://info-bw.de/> -

Permanent link:
<https://info-bw.de/faecher:informatik:oberstufe:java:aoc:aco2023:day18:start?rev=1703151049>

Last update: **21.12.2023 09:30**

