Tag 11 - Cosmic Expansion

- Variante 1

Lösungshinweise Teil 1

- Du musst die Eingabedaten NICHT als zweidimensionales Array speichern und eine komplizierte Pfadsuche starten. Es genügt, wenn du für jede Galaxie (Raute) die korrekten (expandierten) Koordinaten speicherst und anschließend in einer verschachtelten Schleife einfach sämtliche Distanzen zwischen allen Koordinaten berechnest.
- Gehe zunächst die gesamte Eingabedatei **Spalte für Spalte** durch und speichere dir, welche Spalten leer sind. Speichere dir dabei direkt die expandierten x-Koordinaten. Bsp.: Wenn die Spalte 1 und 3 leer sind (bei 0 beginnen zu zählen), dann speicherst du dir die 1 (welche selbst 2 Spalten Platz benötigt!) und die 4, weil nach jeder vorkommenden Koordinate alle nachfolgenden 1 nach "rechts" rücken.
- Anschließend gehst du die Eingabedaten **Zeile für Zeile** durch und speicherst dir die Koordinaten alle Galaxien. Zum Speichern einer Galaxie bietet sich ein zweistelliges int-Array an. Um alle Galaxien zu speichern, eine ArrayList<int[]>.
- Speichere dir pro Zeile die y-Koordinate und pro Character die x-Koordinate.
- Wenn eine Zeile ganz leer ist (keine Raute enthält), dann wird y um eins erhöht.
- Speichere dir bei jeder Raute die Koordinaten als int-Array und füge sie der ArrayList hinzu.
- Iteriere abschließend in zwei verschachtelten Schleifen über die Kombination alle Koordinaten und rechne den Abstand der x-Werte + Abstand der y-Werte zur Gesamtsumme dazu.

Lösungsvorschlag

```
public int partOne() {
    // Sammle alle Spalten, die nur Punkte enthalten
    ArrayList<Integer> emptyColumns = new ArrayList<Integer>();
    for (int x = 0; x < inputLines.get(0).length(); x++) {
        boolean columnEmpty = true;
        for (int y = 0; y < inputLines.size(); y++) {
            if (inputLines.get(y).charAt(x) != '.') {
                columnEmpty = false;
                break;
            }
        }
        if (columnEmpty) {
            emptyColumns.add(x + emptyColumns.size());
        }
    }

    ArrayList<int[]> galaxies = new ArrayList<int[]>();

    int y = 0;
    for (String line: inputLines) {
        char[] lineAsChars = line.toCharArray();
```

```
        // prüfe, ob zeile nur '.' enthält
        boolean zeileLeer = true;
        for (char c: lineAsChars) {
            if (c != '.') {
                zeileLeer = false;
                break;
            }
        }
        if (zeileLeer) {
            y++;
        }

        int x = 0;
        for (char c: lineAsChars) {
            if (emptyColumns.contains(x)) {
                x++;
            }
            if (c == '#') {
                galaxies.add(new int[]{x, y});
            }
            x++;
        }

        y++;
    }

    int distances = 0;

    int skip = 1;
    for (int[] g1: galaxies) {
        for (int i = skip; i < galaxies.size(); i++) {
            int[] g2 = galaxies.get(i);
            distances += Math.abs(g1[0] - g2[0]) + Math.abs(g1[1] - g2[1]);
        }
        skip++;
    }


    return distances;
}
```

Lösungshinweise Teil 2

- Nur minimale Änderungen sind nötig gegenüber der Lösung aus Teil 1.
- Zuvor wurde jede leere Zeile/Spalte verdoppelt (+1). Jetzt soll jede leere Zeile/Spalte durch eine Million leere Zeilen/Spalten **ersetzt** werden. Vorsicht an dieser Stelle: die Rechnung ist daher nun nicht `+1000000`', sondern `+999.999`. Diese Änderung muss an allen drei Stellen durchgeführt werden, wo es direkt um die Abstände der leeren Zeilen/Spalten geht.

Lösungsvorschlag

```
public long partTwo() {
    // Sammle alle Spalten, die nur Punkte enthalten
    ArrayList<Integer> emptyColumns = new ArrayList<Integer>();
    for (int x = 0; x < inputLines.get(0).length(); x++) {
        boolean columnEmpty = true;
        for (int y = 0; y < inputLines.size(); y++) {
            if (inputLines.get(y).charAt(x) != '.') {
                columnEmpty = false;
                break;
            }
        }
        if (columnEmpty) {
            emptyColumns.add(x + emptyColumns.size() * 999999);
        }
    }

    ArrayList<long[]> galaxies = new ArrayList<long[]>();

    long y = 0;
    for (String line: inputLines) {
        char[] lineAsChars = line.toCharArray();

        // prüfe, ob zeile nur '.' enthält
        boolean zeileLeer = true;
        for (char c: lineAsChars) {
            if (c != '.') {
                zeileLeer = false;
                break;
            }
        }
        if (zeileLeer) {
            y += 999999;
        }

        int x = 0;
        for (char c: lineAsChars) {
            if (emptyColumns.contains(x)) {
                x += 999999;
            }
            else if (c == '#') {
                galaxies.add(new long[]{x, y});
            }
            x++;
        }

        y++;
    }
```

```
    long distances = 0;

    int skip = 1;
    for (long[] g1: galaxies) {
        for (int i = skip; i < galaxies.size(); i++) {
            long[] g2 = galaxies.get(i);
            distances += Math.abs(g1[0] - g2[0]) + Math.abs(g1[1] - g2[1]);
        }
        skip++;
    }

    return distances;
}
```

From:
https://www.info-bw.de/ -

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:java:aoc:aco2023:day11:start?rev=1702312422

Last update: **11.12.2023 16:33**