Day 12: Garden Groups

Teil 1 lässt sich noch verhältnismäßig gut lösen, wenn man die richtige Vorgehensweise mittels Rekursion erkennt. Für Teil 2 hingegen muss man sich noch zusätzlich einen Kniff überlegen.

Teil 1

Vorgehensweise:

- Übertrage alle Werte in ein **drei**dimensionales char-Array. Die ersten beiden Dimensionen werden für die x- und y-Koordinate verwendet. Die letzte Dimension soll einen Speicherplatz für 2 Werte haben. An deren Index 0 wird der eigentliche char-Wert des Inputs gespeichert, am Index 1 hingegen eine '1', falls das zugehörige Buchstaben-Feld nocht **nicht** bearbeitet wurde (direkt zu Beginn) und später dann '0', falls es bereits bearbeitet wurde.
- Gehe nun in einer verschachtelten Schleife über alle Koordinaten des Feldes. Wenn ein Feld noch nicht bearbeitet wurde (dritte Dimension, zweiter Index = '1'), dann gehts weiter:
  - Setze eine Instanz-(!)-Variable für area auf 0 und ebenso für perimeter.
  - Rufe anschließend eine Rekursionsmethode auf der aktuellen Koordinate auf, die sich in alle 4 Richtungen "ausbreitet" und so das ganze Areal des aktuellen chars erkunden soll.
  - Wenn die Rekursion schließlich vollständig abgeschlossen ist, dann sind die **Instanz**variablen area und perimeter aktualisiert und du kannst den Preis aktualisieren. Anschließend geht es mit dem nächsten äußeren Schleifendurchlauf zur nächsten Koordinate die noch NICHT bearbeitet wurde.
- Die Rekursion benötigt 3 Parameter:
  1. Der char, um den es geht
  2. Die x-Koordinate
  3. Die y-Koordinate
- Die Rekursion kennt folgende Basisfälle:
  1. Die Koordinaten liegen außerhalb des Felds. Gleichzeitig bedeutet das auch, dass man sich beim Aufruf dieser Rekursion erst in das "Aus" bewegt hat. Man hat also einen Zaun / perimeter mehr gefunden!
  2. Die aktuelle Koordinate ist schon bearbeitet ('0')
  3. Die aktuelle Koordinate enthält nicht den gesuchten char
- Sind die Basisfälle überwunden, dann hat man ein korrektes Feld gefunden, kann also die *area* um eins erhöhen und die aktuelle Koordinate auf "bearbeitet" ('0') setzen.
- Anschließend muss man für alle 4 Richtungen prüfen, ob dort ein Feld mit einem anderen char liegt, dann muss *perimeter* um eins erhöht werden.
- Zum Abschluss muss man die Rekursion auf allen 4 benachbarten Koordinaten fortsetzen.

Lösungsvorschlag

```
public void partOne() {
    // speichere die Höhe und Breite
    width = inputLines.get(0).length();
    height = inputLines.size();

    // Erstelle ein Array, das pro Koordinate den Char speichert
```

```
    // UND zusätzlich, ob die Koordinate bereits besucht war, oder gerade
neu gefunden wurde
    farm = new char[width][height][2];

    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < width; x++) {
            farm[x][y][0] = line.charAt(x);
            farm[x][y][1] = '1'; // "neu gefunden"
        }
    }

    int price = 0;
    // Schaue pro Koordinate, ob dieser char schon bearbeitet wurde
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            if (farm[x][y][1] != '0') {
                area = 0;
                perimeter = 0;

                // Rekursion, zum Abklappern des gesamten Gebiets desselben
chars
                findRegion(farm[x][y][0], x, y);

                price += area*perimeter;
            }
        }
    }

    System.out.println(price);
}

private void findRegion(char c, int x, int y) {
    // Basisfall: Außerhalb des Feldes
    if (x < 0 || x >= width || y < 0 || y >= height) {
        perimeter++;
        return;
    }

    // Basisfall: Bereits bearbeitet ODER falscher char
    if (farm[x][y][1] == '0' || farm[x][y][0] != c) {
        return;
    }

    // sonst: korrektes Feld gefunden -> area um 1 erhöhen und Feld auf
"bearbeitet" setzen
    area++;
    farm[x][y][1] = '0';
```

```
    // prüfe für alle nachbarfelder, ob dort ein anderer Buchstabe/char ist
-> perimeter um 1 erhöhen
    if (x-1 >= 0 && farm[x-1][y][0] != c) {
        perimeter++;
    }
    if (x+1 < width && farm[x+1][y][0] != c) {
        perimeter++;
    }
    if (y-1 >= 0 && farm[x][y-1][0] != c) {
        perimeter++;
    }
    if (y+1 < width && farm[x][y+1][0] != c) {
        perimeter++;
    }

    // Rekursive Suche in allen Nachbarfeldern fortsetzen
    findRegion(c, x-1, y);
    findRegion(c, x+1, y);
    findRegion(c, x, y-1);
    findRegion(c, x, y+1);
}
```

Teil 2

Grundidee: Wir verfahren ganz ähnlich wie in Teil 1, markieren aber im rekursiven Aufruf alle Felder einer Region als "gerade markiert" ('2'). Direkt nach der Rekursion gehen wir über das gesamte Feld und suchen an den Spalten und Zeilen die Grenzübergänge von den "gerade markierten" Feldern zu den Nachbarfeldern. (Eine ziemlich "unschöne", aber funktionale Lösung.)

Lösungsvorschlag

```
public void partTwo() {
    width = inputLines.get(0).length();
    height = inputLines.size();
    farm = new char[width][height][2];
    // 2. Char in dritter Dimension:
    //   0: bereits besucht/bearbeitet
    //   1: neu
    //   2: gerade markiert

    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < width; x++) {
            farm[x][y][0] = line.charAt(x);
            farm[x][y][1] = '1';
        }
    }

    int price = 0;
```

```
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            if (farm[x][y][1] == '1') {
                area = 0;
                perimeter = 0;

                //findRegion2() funktioniert fast gleich wie zuvor: bloß der
perimeter-Teil ist hier nicht mehr nötig
                findRegion2(farm[x][y][0], x, y);

                // NEU: zähle nun alle durchgehenden Kanten an denjenigen
Feldern,
                // die innerhalb von findRegion2() als '2' ("gerade
markiert") gekennzeichnet wurden
                perimeter = countFences();

                price += area*perimeter;
            }
        }
    }

    System.out.println(price);
}

// "Hardcoding": läuft alle horizontalen und vertikalen Kanten zwischen den
Spalten/Zeilen
// UND am Rand entlang, um zusammenhängende Grenzen zu erfassen.
private int countFences() {
    int counter = 0;

    // oberer Rand
    boolean counting = false;
    for (int x = 0; x < width; x++) {
        if (farm[x][0][1] == '2') {
            counting = true;
        } else if (counting) {
            counter++;
            counting = false;
        }
    }
    if (counting) {
        counter++;
    }

    //mittlere Zeilen, jeweils zu den oberen nachbarn
    for (int y = 1; y < height; y++) {
        // zähle Grenzen nach OBEN
        counting = false;
        for (int x = 0; x < width; x++) {
```

```
            if (farm[x][y][1] == '2' && farm[x][y][0] != farm[x][y-1][0]) {
                counting = true;
            } else if (counting) {
                counter++;
                counting = false;
            }
        }
        if (counting) {
            counter++;
        }
    }

    //mittlere Zeilen, jeweils zu den unteren nachbarn
    for (int y = 0; y < height-1; y++) {
        // zähle Grenzen nach UNTEN
        counting = false;
        for (int x = 0; x < width; x++) {
            if (farm[x][y][1] == '2' && farm[x][y][0] != farm[x][y+1][0]) {
                counting = true;
            } else if (counting) {
                counter++;
                counting = false;
            }
        }
        if (counting) {
            counter++;
        }
    }

    // unterer Rand
    counting = false;
    for (int x = 0; x < width; x++) {
        if (farm[x][height-1][1] == '2') {
            counting = true;
        } else if (counting) {
            counter++;
            counting = false;
        }
    }
    if (counting) {
        counter++;
    }


    // linker Rand
    counting = false;
    for (int y = 0; y < height; y++) {
        if (farm[0][y][1] == '2') {
            counting = true;
        } else if (counting) {
            counter++;
```

```
            counting = false;
        }
    }
    if (counting) {
        counter++;
    }

    //mittlere spalten, jeweils zu den linken nachbarn
    for (int x = 1; x < width; x++) {
        // zähle Grenzen nach LINKS
        counting = false;
        for (int y = 0; y < height; y++) {
            if (farm[x][y][1] == '2' && farm[x][y][0] != farm[x-1][y][0]) {
                counting = true;
            } else if (counting) {
                counter++;
                counting = false;
            }
        }
        if (counting) {
            counter++;
        }
    }

    //mittlere spalten, jeweils zu den rechten nachbarn
    for (int x = 0; x < width-1; x++) {
        // zähle Grenzen nach RECHTS
        counting = false;
        for (int y = 0; y < height; y++) {
            if (farm[x][y][1] == '2' && farm[x][y][0] != farm[x+1][y][0]) {
                counting = true;
            } else if (counting) {
                counter++;
                counting = false;
            }
        }
        if (counting) {
            counter++;
        }
    }

    // rechter Rand
    counting = false;
    for (int y = 0; y < height; y++) {
        if (farm[width-1][y][1] == '2') {
            counting = true;
        } else if (counting) {
            counter++;
            counting = false;
```

```
        }
    }
    if (counting) {
        counter++;
    }

    // setze '2' auf '0'
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            if (farm[x][y][1] == '2') {
                farm[x][y][1] = '0';
            }
        }
    }

    return counter;
}

private void findRegion2(char c, int x, int y) {
    if (x < 0 || x >= width || y < 0 || y >= height) {
        return;
    }

    if (farm[x][y][1] != '1' || farm[x][y][0] != c) {
        return;
    }

    area++;
    farm[x][y][1] = '2';

    findRegion2(c, x-1, y);
    findRegion2(c, x+1, y);
    findRegion2(c, x, y-1);
    findRegion2(c, x, y+1);
}
```

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

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:java:aoc:aoc2024:day12:start

Last update: **12.12.2024 15:11**