

Day 9: Disk Fragmenter

Der erste Teil ist ziemlich einfach und gleichzeitig eine wunderbare Übung zum Umgang mit Arrays bzw. ArrayLists und Schleifen! Teil zwei ist hingegen einiges kniffliger und unübersichtlicher zu programmieren.

Teil 1

Wichtige Java-Befehle:

- `ArrayList<Integer> disk = new ArrayList();`: Zum Erstellen der ArrayList, um darin alle Werte der Disk zu speichern.
- `disk.size()`: Um die Länge der ArrayList zu ermitteln. Äquivalent zu `arrayName.length` bei Arrays.
- `for (char c: inputLines.get(0).toCharArray()) {...}`: Zum Iterieren über jeden einzelnen char vom Input
- `int number = (int)(c - '0');`: Alle Zahlen des Inputs sind als char gespeichert. Ein direktes Type-Casting von char in int (`(int)c`) würde dazu führen, dass man den ASCII-Wert der Zahl präsentiert bekommt. Im Fall von '4' z. B. 52! Wenn man nun zusätzlich immer noch den ASCII-Wert des chars '0' (=ASCII 48) abzieht, dann bekommt man die eigentliche Zahl präsentiert: $52 - 48 = 4$
- `disk.get(p)`: Um den Wert am Index p zu erhalten.
- `disk.add(a)`: Um einen Wert a hinten an die ArrayList/Disk anzufügen.
- `disk.set(a, b)`: Um den Wert an der Position a auf b zu setzen. b ersetzt also den vorherigen Wert an Position a.

Vorgehensweise:

- Speichere der Reihe nach alle Dateien in einer `ArrayList<Integer>`. Das Ergebnis muss exakt so aussehen, wie es im Wiki in den längeren Zeilen zu sehen ist. Weil du nur Integer speicherst, empfiehlt es sich, an Leerstellen eine -1 zu speichern.
 - Speichere dir in einer boolean, ob die nächste Ziffer vom Input für eine Datei oder eine Leerstelle steht und switche den boolean zum Abschluss immer hin und her (`leerstelle = !leerstelle`).
 - Füge für jede Ziffer so viele Male die nächst größere Dateien-Ziffer oder eine -1 (Leerstelle) zur Disk hinzu, wie es die aktuelle Ziffer angibt.
- Nutze zwei Zeiger-Variablen:
 1. Eine, die immer auf die vorderste, aktuell vorhandene Leerstelle zeigt.
 2. Eine, die immer auf die hinterste Dateien-Ziffer (Ziffer != -1) zeigt.
- Solange die zwei Variablen sich noch nicht gekreuzt haben (= "Es gibt noch Leerstellen vor einer Datei"):
 - Setze die hintere Ziffer an die vordere Leerstelle (mache anschließend die ursprüngliche, hintere Position der Ziffer zu einer Leerstelle!)
 - Bewege die vordere Zeiger-Variable an die nächste korrekte Position weiter hinten
 - Bewege die hintere Zeiger-Variable an die nächste korrekte Position weiter vorne
- Berechne die Checksumme - das musst du auch selbst hinbekommen!

Lösungsvorschlag

```
public void partOne() {
    ArrayList<Integer> disk = new ArrayList(); // Speichere hierin alle
    Zahlen der Disk

    boolean isFile = true; // Zum Hin- und Herschalten zwischen Datei und
    Leerstelle
    int fileCounter = 0; // Zählt hoch, für späteres 0..111..2222.33. etc.
    der Dateiblöcke

    // Iteriere über jeden Character
    for (char c: inputLines.get(0).toCharArray()) {
        int number = (int)(c-'0'); // Wandle damit die char-Ziffer in
        dieselbe int-Ziffer um

        // Jeweils abwechselnd kommt eine Datei/Leerstelle
        if (isFile) {
            // speichere so viele Mal den fileCounter wie lang die Datei
            sein soll
            for (int i = 0; i < number; i++) {
                disk.add(fileCounter);
            }
            fileCounter++;
        } else {
            for (int i = 0; i < number; i++) {
                disk.add(-1); // speichere -1 für eine Leerstelle
            }
        }
        isFile = !isFile;
    }

    int free = 0; // speichere darin die vorderste Leerstelle
    while (disk.get(free) != -1) { // bewege "free" zum Beginn der ersten
    Leerstelle
        free++;
    }

    int end = disk.size()-1; // speichere damit die hinterste Ziffer die
    KEINE Leerstelle ist
    while (disk.get(end) == -1) {
        end--;
    }

    // solange sich noch eine Leerstelle VOR der hintersten Zahl befindet
    while (free < end) {
        disk.set(free, disk.get(end)); // setze die hinterste Ziffer an die
        vorderste Leerstelle
        disk.set(end, -1); // setze die hinterste Ziffer auf "Leerstelle"
    }
}
```

```
// bewege free zur neuen/nächsten vordersten Leerstelle
while (disk.get(free) != -1) {
    free++;
}

// bewege end zur nächsten/hintersten Ziffer
while (disk.get(end) == -1) {
    end--;
}

// Berechnung der Checksum
long checksum = 0;
for (int i = 0; i < disk.size(); i++) {
    if (disk.get(i) != -1) {
        checksum += i * disk.get(i);
    }
}

System.out.println(checksum);
}
```

Teil 2

Teil 2 ist insgesamt knifflig zu programmieren, da man (zumindest bei der hier gewählten Vorgehensweise) sowohl Buch halten muss über die aktuell vorhandenen Leerstellen, als auch ganze Zahlenblöcke betrachten muss. Die Vorgehensweise lässt sich nicht sinnvoll beschreiben und auch im Code nur schwer nachvollziehen. Der Vollständigkeit halber ist der Code hier trotzdem präsentiert.

Lösungsvorschlag

```
public void partTwo() {
    ArrayList<Integer> disk = new ArrayList(); // Speichere hierin alle
    Zahlen der Disk

    boolean isFile = true; // Zum Hin- und Herschalten zwischen Datei und
    Leerstelle
    int fileCounter = 0; // Zählt hoch, für späteres 0..111..2222.33. etc.
    der Dateiblöcke

    // Iteriere über jeden Character
    for (char c: inputLines.get(0).toCharArray()) {
        int number = (int)(c-'0'); // Wandle damit die char-Ziffer in
        dieselbe int-Ziffer um

        // Jeweils abwechselnd kommt eine Datei/Leerstelle
        if (isFile) {
            // speichere so viele Mal den fileCounter wie lang die Datei
        }
    }
}
```

sein soll

```
        for (int i = 0; i < number; i++) {
            disk.add(fileCounter);
        }
        fileCounter++;
    } else {
        for (int i = 0; i < number; i++) {
            disk.add(-1); // speichere -1 für eine Leerstelle
        }
    }
    isFile = !isFile;
}
```

// Erstelle ArrayList, das alle freie Lücken (jeweils Beginn und Ende) speichert

```
ArrayList<int[]> spaces = new ArrayList();
int start = -1;
int end = -1;
for (int i = 0; i < disk.size(); i++) {
    if (disk.get(i) == -1 && start == -1) {
        start = i;
    }
    else if (disk.get(i) != -1 && start != -1) {
        end = i-1;
        spaces.add(new int[]{start, end});

        start = -1;
        end = -1;
    }
}
```

// Suche von hinten komplette (!) Zahlenblöcke

```
int number = -1;
int length = 0;
for (int i = disk.size()-2; i >= 0; i--) {
    // Falls Beginn eines Blocks bei i+1
    if (!disk.get(i).equals(disk.get(i+1)) && disk.get(i+1) != -1 ) {
        int blockIndex = i+1;
        while (blockIndex < disk.size()-1 &&
disk.get(blockIndex).equals(disk.get(blockIndex+1))) {
            blockIndex++;
        }

        int blockLength = blockIndex-i;

        // Finde einen Space, der lang genug ist
        for (int[] space: spaces) {
            if (space[0] < i+1 && space[1]-space[0]+1 >= blockLength) {
                for (int x = 0; x < blockLength; x++) {
```

```
        disk.set(space[0]+x, disk.get(i+1+x));
        disk.set(i+1+x, -1);
    }
    // reduziere die Größe des benutzten Spaces = verschiebe
den Anfang nach hinten
    space[0] += blockLength;
    break;
}
}
}

// checksum
long checksum = 0;
System.out.println(disk.size());
for (int i = 0; i < disk.size(); i++) {
    if (disk.get(i) != -1) {
        checksum += (long)i * (long)disk.get(i);
    }
}
System.out.println();
System.out.println(checksum);
}
```

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

Permanent link:
<https://info-bw.de/faecher:informatik:oberstufe:java:aoc:aoc2024:day09:start>

Last update: **04.01.2025 14:53**

