Dokumentation 42. BWINF 1. Runde Aufgabe 2

Lösungsidee

Die Lösungsidee besteht darin, die gegebene Aufgabe durch Brute-Force-Methode zu lösen, wobei die Blöcke nacheinander platziert werden. Dabei ist es möglich, die Blöcke zu rotieren. Mittels Backtracking werden alle möglichen Platzierungsvarianten der Würfel in der Kiste systematisch durchprobiert. Falls keine Lösung existiert, wird dies ausgegeben. Im Falle der Entdeckung einer Lösung, d. h. die Kiste ist vollständig gefüllt und es liegen keine übrig gebliebenen Blöcke vor, wird diese Lösung ausgegeben. In einigen Fällen lässt sich auch frühzeitig feststellen, dass keine Lösung möglich ist, beispielsweise wenn das Volumen der Kiste nicht mit der Summe der Volumina der Blöcke übereinstimmt.

Umsetzung

Der Algorithmus wird in Java implementiert und nutzt Rekursion. Bei jedem Aufruf werden ein dreidimensionales Array, das den Inhalt der Kiste repräsentiert, und ein Array übergeben, das angibt, welche Würfel bereits verwendet wurden. Die iterative Überprüfung erfolgt in folgender Struktur: verfügbare Blöcke -> mögliche Positionen des Blocks -> zulässige Orientierungen des Blocks. Es wird überprüft, ob der Block ohne Überlappung mit anderen Blöcken in die Kiste passt. Bei Auffinden einer Lösung, d. h. wenn die Kiste vollständig gefüllt ist, wird das vollständige dreidimensionale Array zurückgegeben und formatiert ausgegeben. Zudem wird zu Beginn überprüft, ob das Volumen der Kiste mit der Summe der Volumina der Blöcke übereinstimmt.

Die Implementierung beinhaltet auch Funktionen und Klassen, die den Umgang mit dreidimensionalen Arrays erleichtern. Beispielsweise unterstützt ein Iterator das Traversieren eines mehrdimensionalen Arrays. Die möglichen Orientierungen eines Blocks werden durch eine einfache Implementierung des Heap-Algorithmus generiert.

Beispiele

Vorgegebene Beispiele:

raetsel1.txt

```
Keine Lösung möglich.
Boxvolumen: 27
Blockvolumen: 28
```

raetsel2.txt

```
Keine Lösung möglich.
Boxvolumen: 27
Blockvolumen: 28
```

raetsel3.txt

```
Keine Lösung möglich.
```

raetsel4.txt

```
Keine Lösung möglich.
Boxvolumen: 125
Blockvolumen: 126
```

raetsel5.txt

```
Keine Lösung möglich.
Boxvolumen: 125
Blockvolumen: 126
```

beispielraetsel.txt

```
Ebene 0:
0 0 0
0 0 0
0 0 0

Ebene 1:
2 2 2
3 G 4
3 5 4

Ebene 2:
1 1 1
1 1 1
1 1 1
```

Eigene Beispiele:

test1.txt

```
Ebene 0:
G
```

test2.txt

```
Keine Lösung möglich.
```

test3.txt

```
Ebene 0:
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
```

```
15 16 17 18 19
20 21 22 23 24

Ebene 1:
25 26 27 28 29
30 31 32 33 34
35 36 37 38 39
40 41 42 43 44
45 46 47 48 49

Ebene 2:
50 51 52 53 54
55 56 57 58 59
60 61 G 62 63
64 65 66 67 68
69 70 71 72 73

Ebene 3:
74 75 76 77 78
79 80 81 82 83
84 85 86 87 88
89 90 91 92 93
94 95 96 97 98

Ebene 4:
99 100 101 102 103
104 105 106 107 108
109 110 111 112 113
114 115 116 117 118
119 120 121 122 123
```

Quelltext

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.util.*;

public class Aufgabe2 {

    private int[] boxDimensions;
    private int[] altBoxDimensions;
    private int[][][] box;
    private int[][][] blocks;

    private Aufgabe2(String[] args){
        readInput(args);
        initBox();

        int vBox = boxDimensions[0] * boxDimensions[1] * boxDimensions[2];
        int vBlocks = 0;
        for (int[][] block : blocks) {
            vBlocks += (block[0][0] + 1) * (block[0][1] + 1) * (block[0][2] + 1);
        }

        if (vBox != vBlocks + 1){
            System.out.println("Keine Lösung möglich.");
            System.out.println("Boxvolumen: " + vBox);
            System.out.println("Blockvolumen: " + (vBlocks + 1));
            System.exit(0);
        }
        int[][][] result = findNext(box, new int[blocks.length]);
        if (result == null){
            System.out.println("Keine Lösung möglich.");
            System.exit(0);
        }
        else{
            for (int i = 0; i < result.length; i++) {
                System.out.println("Ebene " + i + ":");
                for (int j = 0; j < result[0].length; j++) {
                    for (int k = 0; k < result[0][0].length; k++) {
                        if (result[i][j][k] == -2){
                            System.out.print("G ");
                        }
                        else {
                            System.out.print(result[i][j][k] + " ");
                        }
                    }
                    System.out.println();
                }
                System.out.println();
            }
        }
    }

    //Find next block to fit
    private int[][][] findNext(int[][][] cBox, int[] cList){
        int[] ccList = cList.clone();
        int[] firstEmpty = new int[3];
        boolean found = false;
        MDIterator mdiBox = new MDIterator(altBoxDimensions);
        while (mdiBox.hasNext()) {
            int[] next = mdiBox.next();
            if (get3dArray(cBox, next) == -1) {
                firstEmpty = next;
                found = true;
```

```
                break;
            }
        }
        if (!found) {
            return cBox;
        }
        for (int i = 0; i < blocks.length; i++) {
            if (ccList[i] == 0) {
                int[] pos;
                MDIterator mdiPos = new MDIterator(firstEmpty, altBoxDimensions);
                while (mdiPos.hasNext()){
                    pos = mdiPos.next();
                    for (int[] orientation : blocks[i]) {
                        int[][][] ccBox = deepCopy(cBox);
                        if (fit(orientation, pos, ccBox)) {
                            fill(orientation, pos, ccBox, i);
                            ccList[i] = 1;
                            int[][][] temp;
                            if ((temp = findNext(ccBox, ccList)) != null){
                                return temp;
                            }
                        }
                    }
                }
            }
        }
        return null;
    }

    //Initialize box with -1
    private void initBox() {
        altBoxDimensions = arraySum(boxDimensions, new int[]{-1, -1, -1});
        MDIterator iterator = new MDIterator(altBoxDimensions);
        while (iterator.hasNext()) {
            int[] next = iterator.next();
            set3dArray(box, next, -1);
        }
        box[boxDimensions[0] / 2][boxDimensions[1] / 2][boxDimensions[2] / 2] = -2;
    }

    //Check if block fits
    private boolean fit(int[] block, int[] position, int[][][] box) {
        int[] max = arraySum(position, block);
        if (max[0] >= boxDimensions[0] || max[1] >= boxDimensions[1] || max[2] >=
boxDimensions[2]) {
            return false;
        }
        MDIterator mdiBlock = new MDIterator(block);
        while (mdiBlock.hasNext()) {
            int[] c = arraySum(position, mdiBlock.next());
            if (get3dArray(box, c) != -1) {
                return false;
            }
        }
        return true;
    }

    //Fill box with block
    private void fill(int[] block, int[] position, int[][][] cBox, int blockId) {
        MDIterator mdiBlock = new MDIterator(block);
        while (mdiBlock.hasNext()) {
            int[] c = arraySum(position, mdiBlock.next());
            set3dArray(cBox, c, blockId);
        }
    }

    //Get value from 3d array
    private int get3dArray(int[][][] array, int[] position) {
        return array[position[0]][position[1]][position[2]];
    }
```

```
    //Set value in 3d array
    private void set3dArray(int[][][] array, int[] position, int value) {
        array[position[0]][position[1]][position[2]] = value;
    }

    //Add two arrays
    private int[] arraySum(int[] a1, int[] a2) {
        int[] result = new int[a1.length];
        for (int i = 0; i < result.length; i++) {
            result[i] = a1[i] + a2[i];
        }
        return result;
    }

    //Deep copy of 3d array
    private int[][][] deepCopy(int[][][] original) {
        int a = original.length;
        int b = original[0].length;
        int c = original[0][0].length;
        int[][][] res = new int[a][b][c];
        for (int i = 0; i < a; i++) {
            for (int j = 0; j < b; j++) {
                for (int k = 0; k < c; k++) {
                    res[i][j][k] = original[i][j][k];
                }
            }
        }
        return res;
    }

    //Read input from file
    private void readInput(String[] args) {
        if (args.length < 1) {
            System.out.println("Syntax: Aufgabe2 <Pfad zur Eingabedatei>");
            System.exit(0);
        }
        File inputFile = new File(args[0]);
        if (!inputFile.exists()) {
            System.out.println("Datei existiert nicht.");
            System.exit(0);
        }
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(Files.newInputStream(inputFile.toPath()), StandardCharsets.UTF_8))) {
            boxDimensions = parseToIntArray(br.readLine());
            box = new int[boxDimensions[0]][boxDimensions[1]][boxDimensions[2]];
            int blockCount = Integer.parseInt(br.readLine());
            blocks = new int[blockCount][][];
            for (int i = 0; i < blockCount; i++) {
                blocks[i] = getOrientations(arraySum(parseToIntArray(br.readLine()), new int[]{-
1, -1, -1}));
            }
        } catch (IOException e) {
            System.out.println("Fehler beim Laden der Eingabedatei");
            System.exit(0);
        }
    }

    //Parse line of numbers to an integer array
    private int[] parseToIntArray(String line) {
        return Arrays.stream(line.replaceAll("\\D+", "")
                        .split("(?!^)"))
                .mapToInt(Integer::parseInt).toArray();
    }

    //Get all possible orientations of a block
    private int[][] getOrientations(int[] dimensions) {
        List<int[]> out = new ArrayList<>();
        int length = dimensions.length;
        int[] stack = new int[length];
        out.add(dimensions.clone());
        int index = 1;
```

```
        while (index < length) {
            if (stack[index] < index) {
                boolean swapped = true;
                int temp;
                if ((index % 2) == 0 && dimensions[0] != dimensions[index]) {
                    temp = dimensions[0];
                    dimensions[0] = dimensions[index];
                    dimensions[index] = temp;
                } else if (dimensions[stack[index]] != dimensions[index]) {
                    temp = dimensions[stack[index]];
                    dimensions[stack[index]] = dimensions[index];
                    dimensions[index] = temp;
                } else {
                    swapped = false;
                }
                if (swapped && out.stream().noneMatch(e -> Arrays.equals(e, dimensions))) {
                    out.add(dimensions.clone());
                }
                stack[index]++;
                index = 1;
            } else {
                stack[index] = 0;
                index++;
            }
        }
        return out.toArray(int[][]::new);
    }

    public static void main(String[] args) {
        new Aufgabe2(args);
    }


    private class Block{
        private List<int[]> orientations;
        private boolean used;

        private Block(List<int[]> orientations){
            this.orientations = orientations;
            this.used = false;
        }
    }

    //Multidimensional iterator
    private class MDIterator implements Iterator<int[]> {

        private int[] startPosition;
        private int[] endPosition;
        private int[] currentPosition;
        private boolean hasNext = true;

        private MDIterator(int[] startPosition, int[] endPosition) {
            this.startPosition = startPosition;
            this.endPosition = endPosition;
            this.currentPosition = startPosition.clone();
        }

        private MDIterator(int[] endPosition) {
            this(new int[]{0, 0, 0}, endPosition);
        }

        @Override
        public boolean hasNext() {
            return hasNext;
        }

        @Override
        public int[] next() {
            int[] res = currentPosition.clone();

            currentPosition[currentPosition.length - 1]++;
```

```
            for (int i = endPosition.length - 1; i > 0; i--) {
                if (currentPosition[i] > endPosition[i]) {
                    currentPosition[i] = 0;
                    currentPosition[i - 1]++;
                }
            }
            if (currentPosition[0] > endPosition[0]) {
                hasNext = false;
            }
            return res;
        }
    }
}
```