33. Bundesweiter Informatikwettbewerb – Aufgabe 1

Team „ByteSector“

Lösungsidee:

Da in dieser Aufgabe zu jedem Zeitpunkt nur die Kombination an Füllständen der Gefäße relevant ist, die Kombinationen durch eine simple Methode, in diesem Falle der Umfüllprozess, verändert werden können, von einem Ausgangszustand durch diese Methode zu einem Zielzustand gelangt werden und dies in möglichst wenigen Schritten geschehen soll, bietet sich hier eine Lösungssuche durch Breitensuche an. Hierbei wird jede Verzweigung in den Ästen der Breitensuche als Objekt der Klasse **Knotenpunkt** gespeichert, welches eine Sammlung an Objekten der Klasse **Gefäß** und eine Sammlung an Unterknoten besitzt. Außerdem wird in jedem Knotenpunkt ein Objekt der Klasse **Schritt** gespeichert, welche den Umfüllprozess, der zur Erstellung dieses Knotenpunktes geführt hat, darstellt und alle für die am Ende ausgegebene Liste an Schritten relevanten Informationen beinhaltet.

Ein Gefäß besitzt die Eigenschaften

- Volumen
- Füllmenge
- Besitzer

und eine Methode, um die gehaltene Flüssigkeit, welche in diesem Falle Wein ist, soweit, wie es die Umstände erlauben, in einen anderes Gefäß umzufüllen.

Ein Schritt, welcher einen Umfüllprozess darstellt, besitzt die Eigenschaften

- Quellgefäß
- Zielgefäß
- Sammlung an Gefäßen mit der zur Zeit nach dem Umfüllprozess aktuellen Füllständen

Die Methode zum Erstellen neuer Knotenpunkte ist den Knotenpunkten beigelegt, welche ihre Unterknoten erstellen bzw. die Anweisung an bereits existierenden Unterknoten weiterleiten. Jede Ausführung dieser Methode erhöht demnach die Tiefe aller Äste der Breitensuche um eins. Stößt ein Knotenpunkt beim Erstellen eines Unterknotens auf eine Kombination an Füllständen, die einen gültigen Zielzustand darstellt, wird diese Meldung die Kette hochgeleitet und in jedem Knotenpunkt der Unterknoten, welcher die Erfolgsmeldung sendete, markiert.

Das steuernde Objekt gehört der Klasse **Main** an. Dieses übernimmt die Ausführung des Lösungsfindungsprozesses, welcher folgendermaßen abläuft:

- Auslesung des Startzustandes aus angegebener Textdatei
- Erstellung des Hauptknotens mit den Daten des Startzustandes
- Bis zum Lösungsprozess bzw. bis zur Entscheidung, dass die Situation nicht lösbar ist, den Hauptknoten, welcher den Befehl an ggf. existierende Unterknoten weitergibt, dazu anweisen, neue Unterknoten zu erstellen
- Ausgeben der Lösung bzw. der Meldung, dass keine Lösung existiert

Umsetzung:

Die Umsetzung erfolgt in Form eines in Java mithilfe der IDE „Eclipse“ geschriebenen Programmes. Die Endversion des Programms liegt als jar-Datei vor, welche über eine Batch-Datei auf einem Windows-System gestartet werden kann. Zur Eingabe der Textdatei, die den Startzustand beinhaltet, wird der Pfad der Textdatei der Batch-Datei als Startparameter übergeben und von dieser an die jar-Datei weitergeleitet. Dieses kann u.A. durch das Drag&Drop-en der Textdatei auf die Batch-Datei erreicht werden. Die Ausführung wurde unter Windows 7 Home Premium SP1 64-Bit getestet.

Wurde eine Lösung gefunden, erfolgt die Ausgabe in Form eines HTML-Dokumentes, in dem die Füllstände der Gefäße zum Zeitpunkt der einzelnen Schritte sowie Quell- und Zielbehälter der einzelnen Umfüllprozesse in einer Tabelle dargestellt werden. Die Konvertierung der gefundenen Lösungsschritte in das HTML-Dokument wird durch ein Objekt der Klasse **Output** durchgeführt.

Zur Optimierung der Breitensuche, die ansonsten schon bei geringen zweistelligen Tiefen enorme Speicherkapazitäten beansprucht, und zum Feststellen der Unlösbarkeit werden keine Unterknoten gespeichert, deren Sammlung eine bereits vorgekommene Kombination an Füllständen aufweist. Zum Vergleich wird bei jeder Erstellung eines Unterknotens die entsprechende Kombination in eine programmweite Liste eingetragen.
Bei jeder Erstellung einer Unterknotens wird gespeichert, dass während der aktuellen Iteration Fortschritt gemacht wurde. Wird eine Iteration beendet, ohne dass mindestens ein Unterknoten erstellt wurde, d.h. ohne dass Fortschritt gemacht wurde, wird das Problem als unlösbar angesehen und der Lösungsfindungsprozess abgebrochen.

Hinweis:
In der Umsetzung wurden die Klassennamen Knotenpunkt, Gefäß und Schritt durch **Knot**, **Container** und **Step** ersetzt.

Einlesen der Textdatei

```
// Leser mit in den Startparametern angegebenem Pfad (args[0]) erstellen
reader = new BufferedReader(new FileReader(args[0]));

// Textdatei zeilenweise auslesen
input[0] = reader.readLine();
input[1] = reader.readLine();
input[2] = reader.readLine();

// Anzahl an Behältern im Besitz von Person A auslesen
int numA = Integer.parseInt(input[0]);

// Größen und Füllmengen trennen
String[] sizes = input[1].split(" ");
String[] fills = input[2].split(" ");

// Behälter erstellen
// Hierbei beachten, dass die Strings in Integer konvertiert werden
for (int i = 0; i < sizes.length; i++){
        col.add(new Container(Integer.parseInt(sizes[i]), Integer.parseInt(fills[i]), (i < numA), i));
}
```

Übermethode zur Lösungsfindung

```
private boolean findSol(){
        // Erstellen des Hauptknotens mit der Anfangssammlung
        // Hierbei enthält der damit verbundene Schritt weder Quell- noch Zielgefäß
        mainKnot = new Knot(col, new Step(null, null, col));

        // Urpsrungszustand der programmweiten Liste an Füllstandkombinationen hinzufügen
```

```
        Knot.foundKnots.add(mainKnot.summarizeCol());

        // Schleife zur Lösungsfindung
        while (true)
                // Hauptknoten anweisen, Unterknoten zu erstellen
                // Gibt der Hauptknoten true zurück, wurde eine Lösung gefunden, ...
                if (mainKnot.goDown()){
                        // ... die Schritte bis zum Lösungsknoten werden gesammelt und ...
                        mainKnot.getSteps();
                        // ... in 'solution' gespeichert
                        solution = Knot.solSteps;
                        // Letztendlich wird der Lösungsfindungsprozess mit positivem Ergebnis beendet
                        return true;
                }
                // Falls in dieser Iteration keine Lösung gefunden wurde, ...
                else {
                        // ... aber auch kein Fortschritt gemacht wurde, gibt es keine Lösung und der Lösungsfindungsprozess wird mit negativem Ergebnis beendet
                        if (!progress){
                                return false;
                        }
                        // ... aber Fortschritt gemacht wurde, progress für die nächste Iteration auf false setzen
                        else
                                progress = false;
                }

}
```

Erstellung von Unterknoten

```
public boolean goDown(){
        // Falls dieser Knotenpunkt noch keine Unterknoten hat, werden neue erstellt
        if (subKnots.isEmpty()){
                // Es werden alle Umfüllprozesse aller Gefäße untereinander bearbeitet
                for (Container con : col)
                        for (Container con2 : col){
                                // Nicht ausführen, falls Quell- und Zielgefäß der selbe sind, der Quellbehälter leer oder der Zielbehälter voll ist
                                if (con != con2 && con.getFill() > 0 && con2.getFill() != con2.getSize()){

                                        // Sammlung an Gefäßen klonen, damit Änderungen sich nicht auf diesen Knoten auswirken
                                        List<Container> _col = new ArrayList<Container>();
                                        for (Container con3 : col)
                                                _col.add(con3.clone());

                                        // Führe die Umfüllung mit den neuen Gefäßen durch, erreiche sie über den selben Index wie die Originialgefäße
                                        _col.get(col.indexOf(con)).transfer(_col.get(col.indexOf(con2)));

                                        // Überprüfe, ob Kombination an Füllständen bereits vorkam
                                        boolean testPassed = true;
                                        String sumCol = summarizeCol(_col);
                                        for (String str : foundKnots){
                                                if (str.equals(sumCol))
                                                        testPassed = false;
                                        }

                                        // Falls Kombination neu:
                                        if (testPassed){
                                                // Festhalten, dass Fortschritt gemacht wurde
```

```
                                                Main.setProgress(true);

                                                // Neue Kombination eintragen
                                                foundKnots.add(sumCol);

                                                // Neuen Knoten mit geklonten Gefäßen erstellen
                                                Knot _knot = new Knot(_col, new Step(con, con2, _col));
                                                subKnots.add(_knot);

                                                // Wenn erstellter Knoten den Zielzustand darstellt, als solcher speichern und die Kette hochfahren
                                                if (_knot.checkState()){
                                                        solKnot = _knot;
                                                        return true;
                                                }
                                        }
                                }
                        }
        }
        // Falls dieser Knotenpunkt bereits Unterknoten besitzt, wird der Befehl weitergeleitet
        else {
                // Alle Unterknoten durchgehen
                for (Knot knot : subKnots)
                        // Befehl weitergeben und falls einer der Unterknoten auf eine Lösung gestoßen ist, diesen als solcher speichern und die Kette weiter hochfahren
                        if (knot.goDown()){
                                solKnot = knot;
                                return true;
                        }
        }
        return false;
}
```

Beschreibung der Ausgabedatei

```
public void writeOutput() throws IOException{
        // Strings für den HTML-Code von Tabellenkopf und -körper erstellen (siehe unten)
        String header = "", steps = "";

        // Tabellenkopf mit IDs der Behälter füllen
        for (int i = 0; i < col.size(); i++){
                // Kopfzellen je nach Besitzer des Behälters einfärben
                String cellBG = "";
                if (col.get(i).getOwner())
                        cellBG = "bgcolor='#7F00FF'";
                else
                        cellBG = "bgcolor='#00FFFF'";

                header += "<th " + cellBG + ">" + (i+1) + "</th>";
        }

        // Tabellenkörper mit Zeilen für alle Schritte füllen
        for (int i = 0; i < solution.size(); i++){
                steps    += "<tr>"
                                        + "      <td><b>Schritt " + i + "</b></td>";
                // Zellen für jeden einzelnen Behälter des aktuellen Schritten erstellen
                for (int j = 0; j < solution.get(i).getAfter().size(); j++){
                        Container con = solution.get(i).getAfter().get(j);
```

```
                        // Falls ein Behälter als Quell- oder Zielbehälter einer Umschüttung ausgewählt wurde, Zelle dieses Behälters rot bzw. grün einfärben
                        String cellBG = "";
                        // Nicht nach Quell- und Zielbehälter suchen, wenn dies der letzte Zustand ist
                        if (i != solution.size() - 1)
                                // Da jeder 'Step' nur eine Sammlung an Behältern speichert, die nach der Umschüttung vorhanden ist, werden die Prüfung auf Quell- und Zielbehälter auf
                                // den nächsten Schritt bezogen
                                if (solution.get(i+1).getSrc() == con)
                                        cellBG = " bgcolor='#FF0000'";
                                else if (solution.get(i+1).getDest() == con)
                                        cellBG = " bgcolor='#00FF00'";

                        // Zelle mit Füllmenge und Volumen des aktuellen Behälters füllen und Einfärbung vornehmen. Ist keine Farbe vorgesehen, ist cellBG leer.
                        steps += "<td" + cellBG + ">" + con.getFill() + " / " + con.getSize() + "</td>";
                }
                steps += "</tr>";
        }

        // Schließlich die zusammengetragenen Daten in die HTML-Datei schreiben
        writer.write(""
                        + "<html>"
                        + "                <body>"
                        + "                        <table border='1'>"
                        + "                                <tr>"
                        + "                                        <th>Behälter</th>"
                        + "                                        " + header
                        + "                                </tr>"
                        + "                                " + steps
                        + "                        </table>"
                        + "                <body>"
                        + "</html>");
}
```

Beispiele:

fair-beispiel1.txt:

Behälter	1	2	3
Schritt 0	8 / 8	0 / 5	0 / 3
Schritt 1	3 / 8	5 / 5	0 / 3
Schritt 2	3 / 8	2 / 5	3 / 3
Schritt 3	6 / 8	2 / 5	0 / 3
Schritt 4	6 / 8	0 / 5	2 / 3
Schritt 5	1 / 8	5 / 5	2 / 3
Schritt 6	1 / 8	4 / 5	3 / 3
Schritt 7	4 / 8	4 / 5	0 / 3

fair-beispiel3.txt:

Behälter	1	2	3	4
Schritt 0	0 / 6	0 / 26	0 / 13	20 / 50
Schritt 1	6 / 6	0 / 26	0 / 13	14 / 50
Schritt 2	0 / 6	6 / 26	0 / 13	14 / 50
Schritt 3	6 / 6	6 / 26	0 / 13	8 / 50
Schritt 4	0 / 6	12 / 26	0 / 13	8 / 50
Schritt 5	6 / 6	12 / 26	0 / 13	2 / 50
Schritt 6	0 / 6	18 / 26	0 / 13	2 / 50
Schritt 7	0 / 6	5 / 26	13 / 13	2 / 50
Schritt 8	6 / 6	5 / 26	7 / 13	2 / 50
Schritt 9	0 / 6	11 / 26	7 / 13	2 / 50
Schritt 10	6 / 6	11 / 26	1 / 13	2 / 50
Schritt 11	0 / 6	17 / 26	1 / 13	2 / 50
Schritt 12	1 / 6	17 / 26	0 / 13	2 / 50
Schritt 13	1 / 6	4 / 26	13 / 13	2 / 50
Schritt 14	6 / 6	4 / 26	8 / 13	2 / 50

eigenbeispiel1.txt

Behälter	1	2	3
Schritt 0	16 / 16	0 / 9	0 / 7
Schritt 1	7 / 16	9 / 9	0 / 7
Schritt 2	7 / 16	2 / 9	7 / 7
Schritt 3	14 / 16	2 / 9	0 / 7
Schritt 4	14 / 16	0 / 9	2 / 7
Schritt 5	5 / 16	9 / 9	2 / 7
Schritt 6	5 / 16	4 / 9	7 / 7
Schritt 7	12 / 16	4 / 9	0 / 7
Schritt 8	12 / 16	0 / 9	4 / 7
Schritt 9	3 / 16	9 / 9	4 / 7
Schritt 10	3 / 16	6 / 9	7 / 7
Schritt 11	10 / 16	6 / 9	0 / 7
Schritt 12	10 / 16	0 / 9	6 / 7
Schritt 13	1 / 16	9 / 9	6 / 7
Schritt 14	1 / 16	8 / 9	7 / 7
Schritt 15	8 / 16	8 / 9	0 / 7

fair-beispiel2.txt wurde als nicht lösbar befunden.