Aufgabe 2: Dreieckspuzzle

Team-ID: 00325

Team-Name: pwned

Bearbeiter/-innen dieser Aufgabe:
Anton Ferdinand Althoff

23. Oktober 2020

Inhaltsverzeichnis

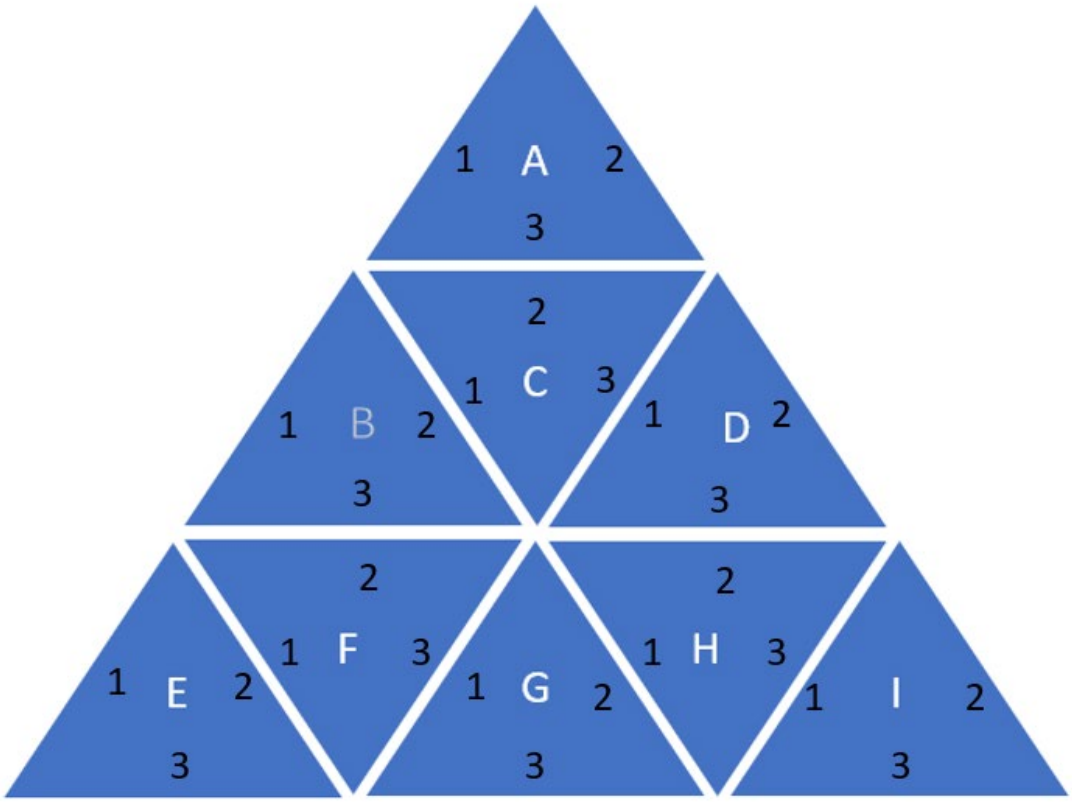
Lösungsidee

Ich habe mich entschieden, das Problem mit Backtracking zu lösen.

Allgemein

Hier haben wir 9 Dreiecke, wobei jedes Dreieck drei Kanten besitzt, mit je einer Ober oder Unterhälfte von einer Figur pro Seite, die jeweils durch die Positive/Negative Variante einer Zahl dargestellt werden (z.B. -4 und 4 sind die Unter und Oberhälfte einer Figur und gehören somit zusammen). Diese Dreiecke müssen so zusammengesetzt werden, dass an jeder Stelle, wo zwei Kanten zusammenkommen, eine Figur entsteht.

Enumeration und Bedingungen



Zuerst enumerieren wir jede Position, die eingenommen werden kann mit den Buchstaben von A bis I. Für jede Position definieren wir auch drei Kanten mit den Zahlen 1-3 (Siehe Abb. 1). Jetzt können wir für jede Position Bedingungen aufstellen, die von dem eingesetzten Dreieck erfüllt werden müssen. Wenn wir bei Position A anfangen, und bis I durchgehen, lauten diese Bedingungen:

A. Keine Bedingungen für A, da wir dieses zuerst einsetzen

B. Keine Bedingungen für B, da es keine Kante von A berührt

C. Zwei Bedingungen müssen erfüllt werden:

   1. Kante 2 von C muss auf Kante 2 von B passen (C.s1 == -1 * B.s2)
   2. Kante 2 von C muss auf Kante 3 von A passen (C.s2 == -1 * A.s3)

D. Eine Bedingung muss erfüllt werden:

   1. Kante 1 von D muss auf Kante 3 von C passen (D.s1 == -1 * C.s3)

E. Keine Bedingung für E, da keine Kante der Dreiecke von A-D berührt werden

F. Zwei Bedingungen müssen erfüllt werden:

   1. Kante 2 von F muss auf Kante 3 von B passen (F.s2 == -1 * B.s3)
   2. Kante 1 von F muss auf Kante 2 von E passen (F.s1 == -1 * E.s2)

G. Eine Bedingung muss erfüllt werden:

   1. Kante 1 von G muss auf Kante 3 von F passen (G.s1 == -1 * F.s3)

H. Zwei Bedingungen müssen erfüllt werden:
    1. Kante 2 von H muss auf Kante 3 von D passen (H.s2 == -1 * D.s3)
    2. Kante 1 von H muss auf Kante 2 von G passen (H.s1 == -1 * G.s2)

I. Eine Bedingung muss erfüllt werden:
    1. Kante 1 von I muss auf Kante 3 von H passen (I.s1 == -1 * H.s3)

(Vergleiche mit Abbildung 1)

Lösung

Diese Bedingung bestimmen, ob ein Dreieck in eine bestimmte Position passt. Jetzt erstellen wir aus den neun Dreiecken, die wir einsetzen dürfen, einen Pool, woraus die einsetzten Dreiecke entnommen werden. Dann gehen wir die Positionen von A bis I durch, und probieren für jede Position die Dreiecke aus dem Pool aus. Sobald wir ein passendes Finden speichern wir es für die Position ab und entfernen es vom Pool von verfügbaren Dreiecken. Dann rücken wir an die nächste Position. Hier machen wir das Gleiche. Dies machen wir, bis wir (1) bei Position I angekommen sind, und das letzte Dreieck an die Position I passt, dann haben wir das Puzzle gelöst, oder (2) wir geraten an einen Punkt, wo wir keine Dreiecke im Pool haben, welche die Bedingung von der jetzigen Position erfüllen. Dann „Backtracken“ wir: wir gehen so weit zurück, bis wir einen Alternativen Lösungsweg für die vorherigen Positionen finden. Konkret gehen wir zuerst zurück an die vorherige Position, und überprüfen, ob auch ein anderes Dreieck die Bedingungen erfüllt. Möglichkeit (1): es gibt eine Alternative, dann speichern wir dieses „neue“ passende Dreieck ab und geben das „alte“ wieder in den Pool, oder (2) es gibt für die vorherige Position keine alternative, dann gehen wir zu dessen Vorgänger usw., bis wir ein anderes Dreieck einsetzen können. Dann geht es wieder zur nächsten Position. Dieser Prozess geht weiter, bis wir (1) eine passende Kombination der Dreiecke gefunden haben, oder (2) wir probieren an Position A alle Dreiecke im Pool durch, und gelangen nie ans Ende; dann gibt es keine Lösung. Diesen Vorgang kann man sich so vorstellen, als würde man alle möglichen Kombinationen der Dreiecke (9! = 362880) in einem Baum-Diagramm darstellen, aber einen Ast abschneiden, sobald man erkennt, dass dieser nicht zu der Lösung führen kann bzw. die Bedingungen nicht erfüllt werden können. (siehe Backtracking Wikipedia).

Umsetzung

Das Programm ist in Java geschrieben, und besteht aus den Klassen Triangle, Solver, SolverInput, und Main. Die Klasse Triangle bietet hier die Basis für die Dreiecks-Objekte. Das Programm fängt mit einem Aufruf von der main() Methode an, welche nur als Entry-Point dient. Diese Erstellt ein neues Objekt SolverInput, welches von STDIN den Dateinamen des Rätsels einliest, und diese öffnet. Aus der Datei werden dann alle Dreiecke eingelesen und in das Array List eingefügt, welches hier unser Pool darstellt. Jetzt erstellt der Konstruktor des SolverInput Objektes eine Solver Objekt, übergibt das Array List, ein leeres Array wo die Lösung reinkommt und ruft die solve() Methode des Objektes auf. In der Solver Klasse liegt die eigentliche Implementierung der Lösung. Sie besteht im Wesentlichen aus der Vergleichsmethode compare1(), welche überprüft, ob ein übergebenes Dreieck die oben beschriebenen Bedingungen für die nächste Position im Lösungsarray erfüllt, eine cloneArray Methode, welche ein Deep Copy (keine Referenzen bzw. Pointer) von einem Array erstellt, und der solve() Methode. In der solve() Methode finden wir die Essenz unseres Programms: hier wird der Pool mit einer For-Loop durchgegangen, und für jedes Dreieck im Pool mithilfe von der compare1() Methode überprüft, ob es in die bisherige Lösung passt. Wenn Ja, fügen wir das Dreieck in das Lösungsarray ein, und dann klonen wir mithilfe von cloneArray() unsere Liste und das Lösungsarray, entfernen von dem Klon der Liste unser passendes Dreieck (Entfernen vom Pool), und erstellen ein neues Solver Objekt. Diesem neuen Objekt geben wir die geklonten Arrays über, und rufen dann die solve() Methode des Objekts aus. Wenn dieser Aufruf TRUE zurückgibt, dann haben wir das Ende erreicht, da das letzte erstelle Solver Objekt eine Leere Liste übergeben bekommen hat (Siehe solve() Quellcode: „ if(list.size() == 0) … “) und die Lösung ausgedruckt hat. Wenn der Aufruf von Solve() FALSE zurückgibt, dann hat das neue Objekt alle Möglichkeiten durchprobiert, und keine

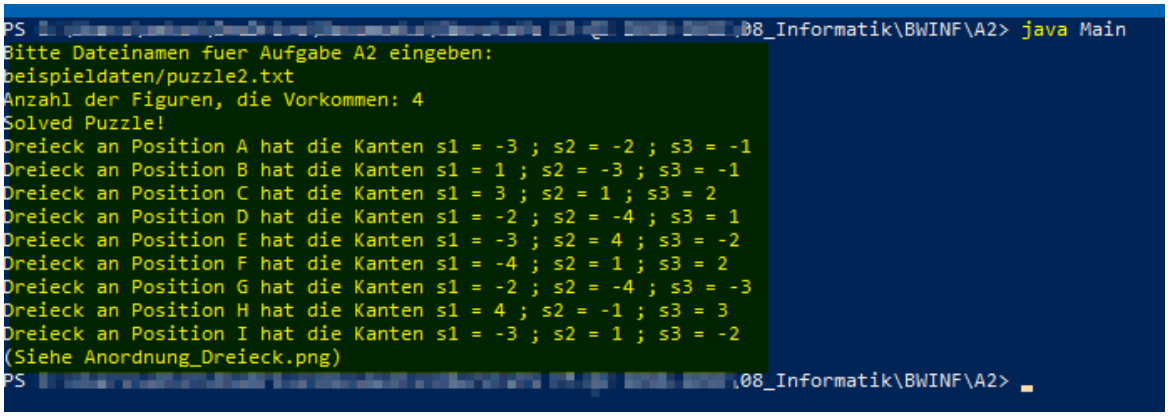
Lösung gefunden, dann müssen wir „Backtracken“, d. h. wie entfernen von der Lösung das letzte Hinzugefügte und probieren weiter durch, bis wir entweder eine Lösung finden, oder wir an Position A alles durchprobiert haben, und dann der erste Solve() Aufruf FALSE zurückgibt, und wir keine Lösung gefunden haben. Im Quellcode stehen weiter detaillierte Kommentare über die Umsetzung.

Beispiele

0. Puzzle0.txt

1. Puzzle1.txt

2. Puzzle2.txt



3. Puzzle3.txt

```
PS [redacted]08_Informatik\BWINF\A2> java Main
Bitte Dateinamen fuer Aufgabe A2 eingeben:
beispieldaten/puzzle3.txt
Anzahl der Figuren, die Vorkommen: 10
Solved Puzzle!
Dreieck an Position A hat die Kanten s1 = 10 ; s2 = 10 ; s3 = 4
Dreieck an Position B hat die Kanten s1 = 10 ; s2 = -3 ; s3 = 2
Dreieck an Position C hat die Kanten s1 = 3 ; s2 = -4 ; s3 = 2
Dreieck an Position D hat die Kanten s1 = -2 ; s2 = 10 ; s3 = 7
Dreieck an Position E hat die Kanten s1 = 10 ; s2 = -5 ; s3 = 10
Dreieck an Position F hat die Kanten s1 = 5 ; s2 = -2 ; s3 = 6
Dreieck an Position G hat die Kanten s1 = -6 ; s2 = -8 ; s3 = 10
Dreieck an Position H hat die Kanten s1 = 8 ; s2 = -7 ; s3 = 9
Dreieck an Position I hat die Kanten s1 = -9 ; s2 = 10 ; s3 = 10
(Siehe Anordnung_Dreieck.png)
PS [redacted]08_Informatik\BWINF\A2>
```

Quellcode

Solver.solve(): boolean

```
public boolean solve() throws Exception
{
    for(int i = 0; i < list.size(); i++) /* Hier gehen wir den Übergebenen Pool durch */
    {
        for(int x = 0; x < 3; x++) /* Da jedes Dreieck drei Seiten hat, daher 3x gedreht werden muss */
        {
            if(compare1(solution, list.get(i))) /* Schauen, ob das Jetzige Dreieck die Bedingungen erfüllt */
            {
                /* Wenn wir hier ankommen, haben wir ein passendes Dreieck für die Position gefunden */
                if(DEBUG)System.out.println("Match( i = " + i + " )  in DEPTH = " + depth + ", SPAWNING NEXTNODE");

                /* Hier klonen wir den Pool und die Lösung */
                ArrayList<Triangle> l2 = cloneArray(list);
                l2.remove(i);

                solution.add(list.get(i));
                ArrayList<Triangle> s2 = cloneArray(solution);

                /* Erstellen ein neues Objekt Solver und übergeben die geklonten Listen */
                Solver nextNode = new Solver(l2,s2,depth+1);
                if(nextNode.solve())
                {
                    /* Das letzte erstellte Objekt hat die Lösung gefunden und ausgedruckt */
                    return true;
                } else
                {
                    /*
                     * Solver Objekt nextNode hat alle Möglichkeiten ausprobiert, und keine Lösung gefunden
                     * daher müssen wir das jetzige Dreieck wieder aus der Liste entfernen.
                     */
                    if(DEBUG)System.out.println("spawned Node Exhausted all Options, on " + depth + " with i = " + i + " , TRYING OTHER");
                    solution.remove(solution.size()-1);
                }
            }
            list.get(i).rotate(); //Dreieck drehen, damit alle Möglichkeiten durchprobiert werden
        }
    }
    if(list.size() == 0)
    {
        /*
         * Wir haben die Lösung gefunden!
         */
        System.out.println("Solved Puzzle!");
        for(int z = 0; z < solution.size(); z++){
            solution.get(z).printSides(z);
         }
        return true; //Kaskadiert rekursiv zurück
    }

    /*
     * Wenn wir an dieser Stelle ankommen, haben wir alle Möglichkeiten für die jetzige Position
     * durchprobiert, und haben keine Lösung gefunden, daher müssen wir Backtracken
     */
    if(DEBUG)System.out.println("Exhausted all options on level " + depth + ", BACKTRACKING");
    return false;
}
```

Solver Konstruktor

```
/*
 * Konstruktor, welcher den Pool List, und die BISHERIGE Lösung übergeben bekommt (+ DEBUG Parameter depth)
 */
public Solver(ArrayList<Triangle> l, ArrayList<Triangle> s, int d)
{
    list = l;
    solution = s;
    depth = d;
}
```

Solver.compare1(ArrayList<Triangle> s, Triangle t): boolean

```
/*
 * Compare1() Überprüft ob die Bedingungen, die von einem Dreieck t erfüllt werden müssen, damit es
 * das nächste Dreieck in der Lösung s wird, erfüllt sind.
 * Wenn die benötigten Bedingungen erfüllt sind, dann ist der Rückgabe wert TRUE, sonst FALSE
 *
 * Hier stehen die Zahlen 0-8 für die Buchstaben A-I
 */
public boolean compare1(ArrayList<Triangle> s, Triangle t) throws Exception
{
    if(DEBUG){
        System.out.println("\n\n Comparing Solution with size = " + s.size() + "with Triangle");
        t.printSides(s.size());
    }
    switch(s.size())
    {
        case 0: // Position A
            return true;
        case 1: // Position B
            return true;
        case 2: // Position C
            if( s.get(0).s3 == -1 * t.s2 && s.get(1).s2 == -1 * t.s1 ) return true;
            break;
        case 3: // Position D
            if( s.get(2).s3 == -1 * t.s1 ) return true;
            break;
        case 4: // Position E
            return true;
        case 5: // Position F
            if( s.get(4).s2 == -1 * t.s1 && s.get(1).s3 == -1 * t.s2 ) return true;
            break;
        case 6: // Position G
            if( s.get(5).s3 == -1 * t.s1 ) return true;
            break;
        case 7: // Position H
            if( s.get(6).s2 == -1 * t.s1 && s.get(3).s3 == -1 * t.s2 ) return true;
            break;
        case 8: // Position I
            if(s.get(7).s3 == -1 * t.s1) return true;
            break;
        default: // Kommen mit der Position nicht klar
            throw new Exception("compare() called with s.size() == " + s.size() + " ,wtf?");
    }
    return false;
}
```

Triangle Klasse

```java
import java.util.*;
import java.io.*;

public class Triangle
{
    public int s1, s2, s3; //Alle Kanten des Dreiecks
    private static char[] position_characters = {'A','B','C','D','E','F','G','H','I'};

    /* Methode, um das Dreieck zu drehen */
    public void rotate()
    {
        int tmp = s3;
        s3 = s2;
        s2 = s1;
        s1 = tmp;
    }

    /* Konstruktor */
    public Triangle(int x, int y, int z)
    {
        s1 = x;
        s2 = y;
        s3 = z;
    }

    /* Methode um Lösung auszudrucken */
    public void printSides(int z){
        System.out.println("Dreieck an Position " + position_characters[z] + " hat die Kanten s1 = "+ s1+ " ; s2 = " + s2 + " ; s3 = " + s3);
    }
}
```

SolverInput Klasse (Datei einlesen)

```java
public class SolverInput
{
    private ArrayList<Triangle> list = new ArrayList<Triangle>();
    private Scanner input = new Scanner(System.in);

    /* Input aus STDIN einlesen mit Vortext und als String zurückgeben */
    public String readInput(String prepend) throws IOException
    {
        System.out.println(prepend);
        return input.nextLine();
    }

    /* Dokument nach Dateinamen öffnen und Scanner zurückgeben */
    public Scanner getFileScannerHandle(String filename)
    {
        Scanner reader = null;
        try
        {
            reader = new Scanner(new FileInputStream(filename), "utf-8");
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Cannot open File, maybe doesnt exist?");
            System.exit(1);
        }
        return reader;
    }

    /* Konstruktor, liest Datei ein und startet den Lösungsprozess */
    public SolverInput() throws Exception
    {
        Scanner reader = getFileScannerHandle(readInput("Bitte Dateinamen fuer Aufgabe A2 eingeben:"));
        System.out.println("Anzahl der Figuren, die Vorkommen: " + reader.nextInt());
        if(reader.nextInt() != 9){
            System.out.println("Puzzle Solver only works with 9 Figures!");
            System.exit(1);
        }
        for(int i = 0; i<9; i++){
            list.add(new Triangle(reader.nextInt(), reader.nextInt(), reader.nextInt())); //Dreiecke aus Datei auslesen und in Liste einfügen
        }
        /* Problem Lösen */
        Solver s = new Solver(list,new ArrayList<Triangle>(), 0);
        if(!s.solve()){
            System.out.println("No solution found...");
        }
    }
}
```