Lösung Aufgabe 2 „Dreieckspuzzle“

Lösungsidee

Wir nutzen eine Brute-Force, um mögliche Lösungen des Puzzles auszuprobieren.

Hierfür unterteilen wir ein gelöstes Puzzle zunächst in „Eckteile“ und „Kernteile“:

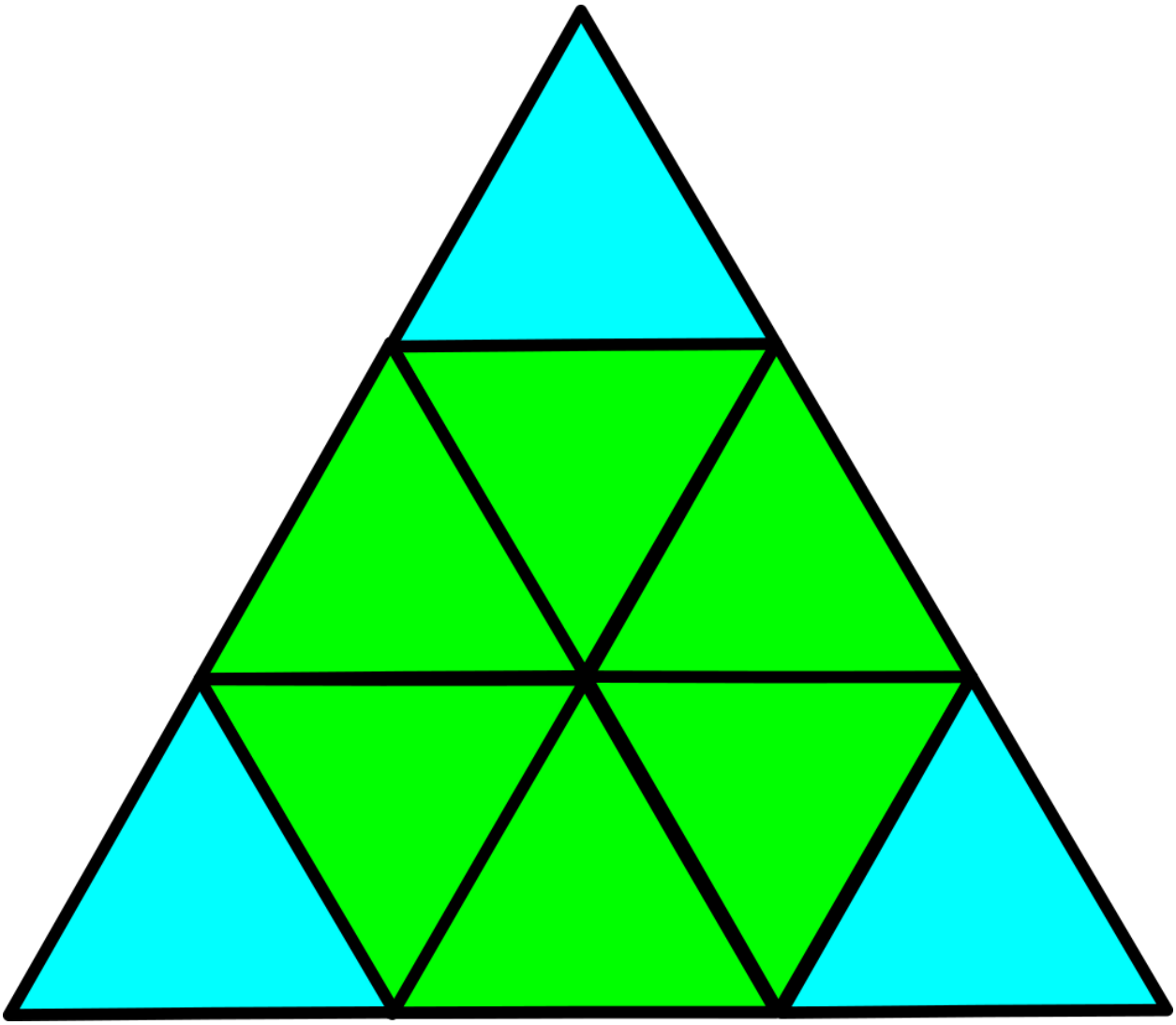

Abbildung 1: Illustration

Den Kern wiederum unterteilen wir in „Randfiguren“ (äußere Figuren des Kerns) und „Kernfiguren“ (innere Figuren des Kerns).

Zunächst probieren wir alle möglichen Kerne aus:

1. Wir beginnen mit einer der drei Seiten eines Teils
2. Wir suchen ein *anderes* Teil, das einen zum dort abgebildeten passenden Figurenteil besitzt

3. Wir bestimmen die gegenüberliegende Seite des passenden Teils, und suchen nun dafür wieder passende Teile (zurück zu Schritt 1)

So lange, bis die 6 Kernteile zusammen kommen. Dann muss noch die letzte Seite des letzten Teils zur ersten Seite des ersten Teils passen, und der Kern ist geschlossen!

Ein geschlossener Kern reicht allerdings noch nicht. Die übrigen Teile müssen noch als Eckteile passen.

Korrektheit

Komplexität

Wir überlegen uns als obere Grenze: Für das erste Kernteil bestehen 9 Wahlmöglichkeiten, für das zweite nur noch 8, usw.

Es gibt also maximal $\frac{9!}{3!} = 60.480$ Möglichkeiten, die Kernteile und ihre Reihenfolge auszuwählen.

Jedes Kernteil kann nun maximal zwei passende Seiten besitzen, also gibt es für jedes Kernteil nochmal zwei Drehmöglichkeiten.

Somit kommt man auf maximal $\frac{9!}{3!} \cdot 2^6 = 60.480 \cdot 64 = 3.870.720$ mögliche Kernanordnungen.

Für jede dieser Kernanordnungen müssen noch Ecken probiert werden. Hierbei gibt es zwei Möglichkeiten, den Kern zu drehen, und für jede $3! = 6$ infragekommende Eckanordnungen.

Schließlich erhält man $\frac{9!}{3!} \cdot 2^6 \cdot 2 \cdot 3! = 9! \cdot 2^7 = 46.448.640$ maximal auszuprobierende Lösungen.

Umsetzung

Implementierung in der modernen und performanten Programmiersprache Go.

Kompilieren

`go build` (erzeugt `a5-Wichteln`) oder `go build main.go` (erzeugt `main`)

Verwendung

`go run main.go <pfad>` oder `./main <pfad>`

Beispiel: `./main beispieldaten/puzzle0.txt`

Ausgabe

```
Teile:
<Teile als ASCII-Art>

Lösung:
<Gelöstes Puzzle als ASCII-Art>

Zeit verstrichen: <Verstrichene Zeit in Millisekunden> ms
```

oder

```
Puzzle unlösbar
Zeit verstrichen: <Verstrichene Zeit in Millisekunden> ms
```

Bibliotheken

- `fmt`: Ausgabe & Formattierung
- `io/ioutil`: Einlesen der Datei
- `os`: Programmargumente
- `strconv`: String/Integer-Konversion
- `strings`: Auftrennen von Text
- `time`: Zeitmessung

Quellcode

main.go

```
1  package main
2
3  import (
4          "fmt"
5          "io/ioutil"
6          "os"
7          "strconv"
8          "strings"
9          "time"
10 )
11
12 // Figur - Ganzzahl von -128 bis 127
13 type Figur int8
14
15 // Teil - Drei Figuren
16 type Teil [3]Figur
17
18 // Kernteil - Seite, Teil, und Vorheriges
19 type Kernteil struct {
20         seite      uint8
21         teil       uint8
22         vorheriges *Kernteil
23 }
24
25 // Eckteil - Seite und Teil
26 type Eckteil struct {
27         seite uint8
28         teil  uint8
```

```
}

// VerwendeteTeile - Flag
type VerwendeteTeile uint16

// Verwendet - Gibt zurück, ob ein Teil verwendet ist
func (teile VerwendeteTeile) Verwendet(teil uint8) bool {
        return teile&(1<<teil) > 0
}

// Verwende - Verwendet ein Teil und gibt Flag zurück
func (teile VerwendeteTeile) Verwende(teil uint8) VerwendeteTeile {
        return teile | (1 << teil)
}

func main() {
        // Zeitmessung
        nanos := time.Now().UnixNano()
        verstricheneZeit := func() {
                fmt.Println("Zeit verstrichen:",
 ↪  float64(time.Now().UnixNano()-nanos)/1e6, "ms")
        }
        // Eingabe
        if len(os.Args) != 2 {
                println("Verwendung: <pfad>")
                return
        }
        text, err := ioutil.ReadFile(os.Args[1])
        if err != nil {
                panic(err)
        }
        lines := strings.Split(string(text), "\n")
        // Teile einlesen
        teile := make([]Teil, 9)
        for l := 2; l < 11; l++ {
                var teil Teil
                figuren := strings.Split(lines[l], " ")
                for f := 0; f < 3; f++ {
                        teilF, _ := strconv.Atoi(figuren[f])
                        teil[f] = Figur(teilF)
                }
                teile[l-2] = teil
        }
        // Rekursive Funktion, die "Kerne" erzeugt und probiert
        var probiereKerne func(Figur, *Kernteil, VerwendeteTeile, uint8)
```

```
73          probiereKerne = func(passendZurStartfigur Figur, vorheriges
    ↪  *Kernteil, verwendeteTeile VerwendeteTeile, anzahlKernteile uint8) {
74                  if anzahlKernteile == 6 {
75                          // Abbruchbedingung: Sechs Kernteile
76                          // Probieren: Erster Test: Passt die Seite des
                            ↪  Kernteils zur ersten Seite (ist der Kern
                            ↪  geschlossen?)
77                          if teile[vorheriges.teil][vorheriges.seite] !=
                            ↪  passendZurStartfigur {
78                                  return
79                          }
80                          // Randfiguren des Kerns
81                          randfiguren := [6]Figur{}
82                          cursor := vorheriges
83                          for i := range randfiguren {
84                                  // Umgekehrte Reihenfolge
85                                  randfiguren[5-i] =
    ↪  teile[cursor.teil][(cursor.seite+2)%3]
86                                  if i < 5 {
87                                          cursor = cursor.vorheriges
88                                  }
89                          }
90                          // Rekursive Funktion, die alle möglichen
                            ↪  Eckanordnungen probiert
91                          var probiereEcken func(uint8, VerwendeteTeile,
                            ↪  []Eckteil)
92                          probiereEcken = func(versatz uint8, verwendeteTeile
    ↪  VerwendeteTeile, ecken []Eckteil) {
93                                  if len(ecken) == 3 {
94                                          if verwendeteTeile != 0b111111111 {
95                                                  // "Sanity-check": Am Ende
                                                    ↪  müssen alle Teile
                                                    ↪  verwendet worden sein
96                                                  panic("Teile mehrfach
                                                    ↪  verwendet")
97                                          }
98                                          // 3 Passende Ecken wurden gefunden,
                                            ↪  das Puzzle ist gelöst!
99                                          // Ausgabe:
100                                         // Obere Figurenteile sind
                                            ↪  Großbuchstaben, untere
                                            ↪  Kleinbuchstaben
101                                         figurenOben, figurenUnten :=
    ↪  [27]rune{}, [27]rune{}
102                                         for f := 0; f < 27-8; f++ {
```

```
                                                figurenOben[f],
    figurenUnten[f] = rune('A'+f), rune('a'+f)
                                        }
                                        // Gibt für eine Figur den Buchstaben zurück
                                        figur := func(num Figur) rune {
                                                if num < 0 {
                                                        return figurenUnten[-num]
                                                }
                                                return figurenOben[num]
                                        }
                                        // Teile ausgeben, platzsparend nebeneinander
                                        fmt.Println("Teile:")
                                        teileFmt := []string{"", "", "", ""}
                                        for _, teil := range teile {
                                                for i, zeile := range []string{
                                                        `   /-\    `,
                                                        fmt.Sprintf(`  /%c  \%c\   `, figur(teil[0]), figur(teil[1])),
                                                        fmt.Sprintf(` /  %c  \  `, figur(teil[2])),
                                                        `/-------\ `,
                                                } {
                                                        teileFmt[i] += zeile
                                                }
                                        }
                                        fmt.Println(strings.Join(teileFmt, "\n"))
                                        fmt.Println()
                                        // Lösung ausgeben
                                        fmt.Println("Lösung:")
                                        format := []rune(`           /-\
          /0 0\
         /  0  \
        /-------\
       / \  3  / \
      /3 4\4 4/4 3\
     /  4  \ /  4  \
    /---------------\
   / \  4  / \  4  / \
  /2 2\3 4/4 4\4 3/1 1\
 /  2  \ /  3  \ /  1  \
```

```
/----------------------\
`)
                                        // Unterteilung in 5 Figurengruppen: Eckfiguren (0-2), Randfiguren des Kerns (3), Kernfiguren (4)
                                        figuren := [5][]Figur{}
                                        // Ecken einsetzen
                                        for i, ecke := range ecken {
                                                teil := teile[ecke.teil]
                                                figuren[i] = []Figur{teil[ecke.seite], teil[(ecke.seite+1)%3], teil[(ecke.seite+2)%3]}
                                        }
                                        if versatz == 0 {
                                                // Kein Versatz: Randfiguren und Kernteile sind in richtiger Reihenfolge
                                                figuren[3] = randfiguren[:]
                                                cursor = vorheriges
                                        } else {
                                                // Versatz von 1: Randfiguren und Kernteile müssen um eins verschoben werden
                                                // Dabei wird das erste Element zum neuen Letzten
                                                figuren[3] = append(randfiguren[1:], randfiguren[0])
                                                // "cursor" ist ein einfach verlinkter Listenknoten, der auf das Erste Element zeigt
                                                // Dessen Nachfolger wird jetzt der Zweite Knoten
                                                cursor.vorheriges = vorheriges
                                        }
                                        // Kernfiguren einsetzen
                                        figuren[4] = make([]Figur, 12)
                                        for i := 5; i >= 0; i-- {
                                                teil := teile[cursor.teil]
                                                // Pro Teil immer Zwei Figuren
                                                figuren[4][i*2] = teil[(cursor.seite+1)%3]
                                                figuren[4][i*2+1] = teil[cursor.seite]
```

```
                cursor = cursor.vorheriges
        }
        // Im String sind die Figurengruppen nicht in der richtigen Reihenfolge:
        // - Die Ecken müssen noch gedreht werden
        // - Die Randfiguren sind kreisförmig angeordnet, und nicht von oben nach unten - links nach rechts
        // - Die Kernfiguren ebenfalls
        reihenfolge := [5][]int{
                // Drehung der Ecken
                {1, 2, 0},
                {0, 1, 2},
                {2, 0, 1},
                // Kreisform Randfiguren
                {0, 5, 1, 4, 2, 3},
                // Kreisform Kernfiguren
                {11, 0, 1, 2, 10, 3, 9, 4, 8, 7, 6, 5},
        }
        // N-tes Element jeder Figurengruppe
        n := [5]int{}
        for i, c := range format {
                // Platzhalter für Figurengruppen sind die jeweiligen Zahlen
                if c >= '0' && c <= '4' {
                        // Zahl 48 - 52 (ASCII) in Zahl 0 - 4 konvertieren, "c" gibt Figurengruppe an
                        c -= '0'
                        // N-te Figur aus der passenden Figurengruppe in der richtigen Reihenfolge
                        format[i] = figur(figuren[c][reihenfolge[c][n[c]]])
```

```
193                                                     // Nächstes Element der Figurengruppe beim nächsten Platzhalter
194                                                     n[c]++
195                                             }
196                                     }
197                                     // Ausgeben
198                                     fmt.Println(string(format))
199                                     verstricheneZeit()
200                                     // Programm beenden
201                                     os.Exit(0)
202                             }
203                             // Passende Figur ist anderer Teil der Randfigur an entsprechender Stelle
204                             passendeFigur := -randfiguren[uint8(2*len(ecken))+versatz]
205                             for teil := uint8(0); teil < 9; teil++ {
206                                     if verwendeteTeile.Verwendet(teil) {
207                                             // Teil schon verwendet
208                                             continue
209                                     }
210                                     for seite, figur := range teile[teil] {
211                                             // Seiten probieren
212                                             if figur == passendeFigur {
213                                                     // Erstelle Kopie der Ecken & füge neue Ecke hinzu
214                                                     eckenKopie := make([]Eckteil, len(ecken)+1)
215                                                     for i, ecke := range ecken {
216                                                             eckenKopie[i] = ecke
217                                                     }
218                                                     eckenKopie[len(ecken)] = Eckteil{uint8(seite), teil}
219                                                     // Probiere weitere Ecken
220                                                     probiereEcken(versatz, verwendeteTeile.Verwende(teil), eckenKopie)
221                                                     break
222                                             }
223                                     }
```

```
224                                 }
225                         }
226                         // Ecken probieren, mit Versatz 0 und 1
227                         probiereEcken(0, verwendeteTeile, []Eckteil{})
228                         probiereEcken(1, verwendeteTeile, []Eckteil{})
229                         return
230                 }
231                 for teil := uint8(0); teil < 9; teil++ {
232                         if verwendeteTeile.Verwendet(teil) {
233                                 // Teil schon verwendet
234                                 continue
235                         }
236                         // Passende Seiten ermitteln
237                         passendeFigur :=
    ↪ -teile[vorheriges.teil][vorheriges.seite]
238                         passendeSeiten := []uint8{}
239                         for seite, figur := range teile[teil] {
240                                 if figur == passendeFigur {
241                                         passendeSeiten =
    ↪ append(passendeSeiten, uint8(seite))
242                                 }
243                         }
244                         // Alle Seiten passen: Seiten sind identisch, Drehung
                             ↪  ist egal
245                         if len(passendeSeiten) == 3 {
246                                 passendeSeiten = []uint8{0}
247                         }
248                         for _, seite := range passendeSeiten {
249                                 // Kernteil erzeugen. Seite ist hierbei die,
                                     ↪  an die das nächste Kernteil anbinden muss.
250                                 kernteil := &Kernteil{(seite + 2) % 3, teil,
    ↪ vorheriges}
251                                 // Neues Kernteil übergeben, Teil verwenden,
                                     ↪  Anzahl Kernteile erhöhen
252                                 probiereKerne(passendZurStartfigur,
    ↪ kernteil, verwendeteTeile.Verwende(teil), anzahlKernteile+1)
253                         }
254                 }
255         }
256         for teil := uint8(0); teil < 5; teil++ {
257                 for seite := uint8(0); seite < 3; seite++ {
258                         // Unter 4 Teilen muss eines dabei sein, dass keine
                             ↪  Ecke, sondern ein Kernteil ist
259                         // Starte Brute-Force mit Kernteil
```

```
                        probiereKerne(-teile[teil][seite], &Kernteil{(seite
 ↪  + 2) % 3, teil, nil}, VerwendeteTeile(0).Verwende(teil), 1)
                        // TODO break
                }
        }
        // Kein Abbruch ist erfolgt: Das Puzzle konnte nicht gelöst werden
        fmt.Println("Puzzle unlösbar")
        verstricheneZeit()
}
```

Beispiele

In `loesungen` als Textdateien mit gleichem Namen wie die Aufgabe.

`puzzle0.txt`

```
Teile:
   /-\       /-\       /-\       /-\       /-\       /-\       /-\       /-\
    ↪  /-\
  /b c\     /C b\     /b c\     /b D\     /C d\     /b D\     /C C\     /d C\
   ↪  /c B\
 /  B  \   /  b  \   /  C  \   /  B  \   /  D  \   /  c  \   /  b  \   /  b  \
  ↪  /  d  \
/-------\ /-------\ /-------\ /-------\ /-------\ /-------\ /-------\ /-------\
 ↪  /-------\ /-------\

Lösung:
           /-\
          /C b\
         /  c  \
        /-------\
       / \  C  / \
      /c B\b b/B b\
     /  b  \ /  D  \
    /---------------\
   / \  B  / \  d  / \
  /b C\c d/D c\C D/d C\
 /  C  \ /  b  \ /  b  \
/-----------------------\

Zeit verstrichen: 0.275944 ms
```

puzzle1.txt

```
Teile:
   /-\       /-\       /-\       /-\       /-\       /-\       /-\       /-\
    ↪  /-\
  /B b\     /C d\     /b b\     /B b\     /d D\     /b D\     /B c\     /b c\
   ↪  /D c\
 /  C  \   /  b  \   /  D  \   /  c  \   /  b  \   /  D  \   /  d  \   /  D  \
  ↪  /  C  \
/-------\ /-------\ /-------\ /-------\ /-------\ /-------\ /-------\
 ↪  /-------\ /-------\

Lösung:
           /-\
          /D b\
         /  c  \
        /-------\
       / \  C  / \
      /b C\c D/d b\
     /  B  \ /  C  \
    /---------------\
   / \  b  / \  c  / \
  /b d\D D/d B\b B/b b\
 /  D  \ /  c  \ /  D  \
/-----------------------\

Zeit verstrichen: 0.321481 ms
```

puzzle2.txt

```
Teile:
   /-\       /-\       /-\       /-\       /-\       /-\       /-\       /-\
    ↪  /-\
  /d c\     /c e\     /B C\     /d b\     /C D\     /d c\     /d B\     /D E\
   ↪  /c d\
 /  b  \   /  B  \   /  e  \   /  B  \   /  B  \   /  e  \   /  c  \   /  b  \
  ↪  /  E  \
/-------\ /-------\ /-------\ /-------\ /-------\ /-------\ /-------\
 ↪  /-------\ /-------\

Lösung:
           /-\
          /e B\
         /  C  \
        /-------\
```

```
       / \  c  / \
      /c b\B e/E c\
     /  d  \ /  d  \
    /---------------\
   / \  D  / \  D  / \
  /B c\C B/b B\b E/e d\
 /  d  \ /  d  \ /  c  \
/-----------------------\

Zeit verstrichen: 0.386294 ms
```

puzzle3.txt

```
Teile:
   /-\       /-\       /-\       /-\       /-\       /-\       /-\       /-\
    ↪  /-\
  /K K\     /J I\     /K f\     /c K\     /G F\     /C D\     /g i\     /j K\
   ↪  /K d\
 /  E  \   /  h  \   /  K  \   /  H  \   /  c  \   /  e  \   /  K  \   /  K  \
  ↪  /  C  \
/-------\ /-------\ /-------\ /-------\ /-------\ /-------\ /-------\
 ↪  /-------\ /-------\

Lösung:
           /-\
          /K K\
         /  j  \
        /-------\
       / \  J  / \
      /K H\h I/i K\
     /  c  \ /  g  \
    /---------------\
   / \  C  / \  G  / \
  /K E\e D/d C\c F/f K\
 /  K  \ /  K  \ /  K  \
/-----------------------\

Zeit verstrichen: 0.272592 ms
```

puzzle4.txt

Zusätzliches Beispiel:

```
10
9
10 -10 4
```

```
9 8 -7
10 -5 10
-2 10 7
6 5 -2
2 3 -4
-6 -8 10
-9 10 10
10 -3 -2

Puzzle unlösbar
Zeit verstrichen: 0.19849 ms
```