Aufgabe 2: Dreieckspuzzle

Team-ID: 00537

Team-Name: LuC++

Bearbeiter dieser Aufgabe:
Lucas Schwebler

23. Oktober 2020

Inhaltsverzeichnis

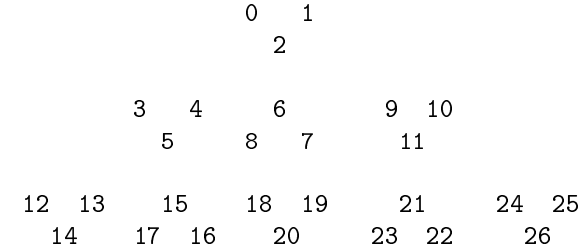
1 Lösungsidee

Bereits die Aufgabenstellung deutet an, dass dieses Problem mit Bruteforce zu lösen ist. So heißt es „*Nach einigem Probieren* hat Lizzy eine Lösung gefunden“. Man könnte naiv alle möglichen Positionen und Rotationen der Teile durchprobieren. Insgesamt sind dies $9! \cdot 3^9 = 7.142.567.040$ Kombinationen. Es wird zwar wahrscheinlich bereits vorher eine Lösung gefunden. Falls diese aber nicht existiert, könnte es etwas dauern das festzustellen...[1]

Daher wird in dieser Lösung ein Backtracking Ansatz verwendet. Die Puzzleteile werden also nacheinander platziert und wenn es irgendwo nicht mehr weiter geht, werden vorherige Entscheidungen verändert anstatt möglicherweise über eine Million Fälle zu überprüfen (naiver Ansatz), die keine Chance auf Erfolg haben, da die ersten Teile bereits nicht zusammen passen.

2 Umsetzung

Um besser mit dem Puzzle arbeiten zu können, wird es werden die Seiten eines Teils so indiziert, dass für Teil k die Seiten die Indizes $3k$, $3k+1$ und $3k+2$ haben (im Uhrzeigersinn):



[1] Auch mit dem Backtrackingansatz kann es etwas dauern, dies festzustellen, falls viele der Teile aneinander passen. Dann muss man möglicherweise auch über eine Minute warten, bis das Programm fertig ist. Die Laufzeit ist und bleibt nämlich exponentiell. Dennoch ist der Ansatz schneller als eine naive Bruteforcelösung.

Nun wird in einem Array *adj* gespeichert, welche Seiten sich berühren:

```
adj[6] = 2
adj[8] = 4
adj[9] = 7
adj[15] = 5
adj[17] = 13
adj[18] = 16
adj[21] = 11
adj[23] = 19
adj[24] = 22
```

Für alle Seiten, die zu keiner anderen adjazent sind, ist der Wert -1. Da beim Backtracking nur Konflikte mit bereits platzierten Teilen relevant sind, wird immer für den größeren Index der kleinere gespeichert. So ist z.B. $adj[2] = -1 \wedge adj[6] = 2$.

Beim Lösen wird immer ein Teil gewählt, das noch nicht verwendet wurde. Dann werden für dieses alle 3 Rotationen getestet. Gibt es keine Konflikte wird das Teil hinzugefügt und die Rekursion fortgesetzt, also wenn für alle Seiten s gilt: $adj[s] = -1 \vee state[adj[s]] = -state[s]$, wobei $state[i]$ der Zustand, bzw. das Symbol der Seite an Index i sei.

Die Ausgabe erfolgt in der selben Anordnung, in der bereits die Indizes abgebildet waren. Bei dieser ist zu beachten, dass die einzelnen Werte nicht für die Ecken stehen, sondern für die Seiten. Durch die Darstellung der Indizes und die Werte des Adjazenzarrays ist die Darstellung aber hoffentlich selbsterklärend.

3 Beispiele

```
puzzle0.txt:
Lösung gefunden:
                                1      -1
                                   -2


                  -1     2          2           1     -1
                     -1         -2     -1          3


      2     2           1          3     -2          -3        -3    2
         -1          -2    -3         -1          2     3         -1



puzzle1.txt:
Lösung gefunden:
                                -1     3
                                   -1


                  -1     2          1           1     -2
                     -3         -2     -1          -3


      -1    -2          3          2     1           3         -3    3
         3           2     -2         -1          -1    3         -1



puzzle2.txt:
Lösung gefunden:
                                -3     -2
                                   -1


                  1      -3         1           -2    -4
                     -1         3      2           1


      -3    4           1          -2    -4          -1        -3    1
         -2          -4    2          -3          4     3         -2
```

```
puzzle3.txt:
Lösung gefunden:
                              10     10
                                 4


                  10     -3         -4          -2    10
                     2          3       2           7


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

4 Quellcode

Lösung in Python:

```
1
2  def solve(cur, state, vis):
3      # Gehe alle Teile des Puzzles durch
4      for i in range(9):
5          # Betrachte das Teil nur, wenn es zuvor noch nicht verwendet wurde (Backtracking)
6          if not vis[i]:
7              # Gehe alle Rotationen des Teils durch
8              for k in range(3):
9                  possible = True
10                 for j in range(3):
11                     # Setze die Indizes (siehe oben) auf die Werte des Teils
12                     # (k + j) % 3 gibt die Position der j-ten Seite bei Rotation k an.
13                     state[3*cur + j] = puzzle[i][(k + j) % 3]
14                     # Gibt es einen Konflikt mit einem bereits platzierten Teil?
15                     if adj[3*cur + j] != -1 and state[3*cur + j] != -state[adj[3*cur + j
       ]]:
16                         possible = False
17                 if possible:
18                     if cur == 8:
19                         # Lösung gefunden
20                         return state
21                     else:
22                         vis[i] = True
23                         # Führe die Suche rekursiv fort
24                         res = solve(cur+1, state, vis)
25                         # Falls dieser Pfad zu einer validen Lösung führt, muss nicht
       weiter gesucht werden
26                         if(res):
27                             return res
28                         vis[i] = False
29     return False
```