Lukas Kesch
lukas.kesch@gmail.com
Teilnahme-ID: 57621
Team-ID: 00579

Aufgabe 2: Dreieckspuzzle

39. Bundeswettbewerb für Informatik - Runde 1 (23.11.2020)

1. Aufgabenstellung

Gegeben ist ein Dreieckspuzzle. Dieses besteht aus neun gleich großen gleichseitigen Dreiecks-Teilen. An jeder Kante eines Teils ist eine Nummer angegeben. Mithilfe aller neun Teile kann ein großes Dreieck gelegt werden. Dabei muss für alle Kanten, die an eine weiteren Innen-Kante grenzen, gelten, dass die Zahl an der einen Kante der negierten Zahl an der anderen Kante entspricht.

2. Lösungsidee

Um ein richtiges Ergebnis garantieren zu können, werden alle möglichen Dreiecks-Anordnungen in Betracht gezogen und es wird überprüft, ob an allen verbindenden Kanten die Zahlen stimmen. Es wird ein rekursiv operierender Backtracking-Algorithmus in Verbindung mit einem Pruning-Ansatz eingesetzt. Dabei wird bei jeder Teiländerung überprüft, ob diese legitim ist. Sollte dies nicht der Fall sein, so wird die aktuelle Konfiguration und alle Konfigurationen, die sich aus dieser ableiten lassen, verworfen.

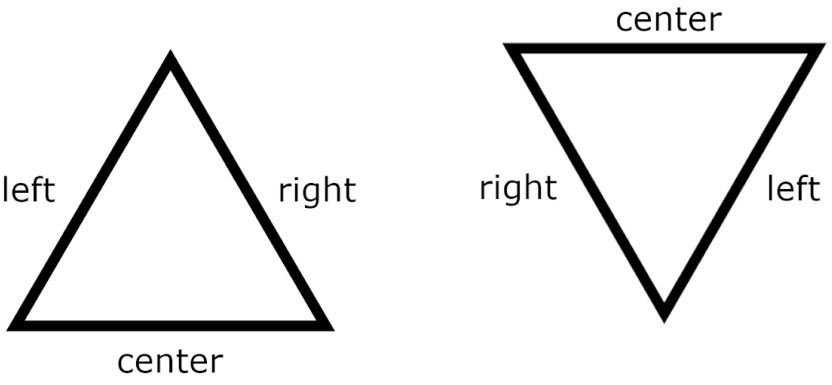
3. Umsetzung

Das Programm wurde in C++ geschrieben und mit dem MSVC Compiler kompiliert. Im Verlauf des Programms wird von der *Standard Libary* ausgiebig gebrauch gemacht, um die Übersichtkichkeit zu bewahren.

Die einzelnen Puzzleteile werden durch die Structure *Piece* repräsentiert. Diese speichert einen *left*, *right*, *center* und *id* int-Wert. Der *id*-Wert wird lediglich für Debugging Zwecke eingeführt und ist somit nicht weiter relevant. Eine Zeile (z.B. 1 3 2) in den Beispieldateien, die ein Puzzlestück angibtt, wird folgendermaßen eingelesen:

Piece.left = 1; *Piece.right* = 3; *Piece.center* = 2;

In dem Puzzle treten die dreieckigen Teile in zwei Fällen auf. Im ersten Fall stehen sie “normal” im zweiten Fall auf dem Kopf (Rotation um 180 Grad). Folgende Abbildung gibt an, wie die *left*, *right* und *center* Attribute vom Dreieck abzulesen sind.



Das Programm startet damit, die neun verschiedenen Puzzelteile einzulesen. Dabei werden gleich die *Piece* Objekte erzeugt und der left, right und center Wert gesetzt. Wichtig ist, dass auch Rotationen des Puzzleteils abgespeichert werden. Deshalb werden zu jedem eingelesenen Puzzleteil drei *Piece* Objekte erstellt.

Nach dem Einlesen beginnt direkt der rekursive Algorithmus, der das Puzzle löst. Es wird ein Puzzleteil auf Index 0 (bildlich gesehen unten links) und eins auf Index 1 (bildlich gesehen rechts vom ersten) gesetzt. Bevor jedoch ein weiteres Puzzleteil gesetzt werden darf, wird überprüft, ob die Teile auf Index 0 und Index 1 zusammenpassen (eventuell in gedrehter Lage). Falls dies zutrifft, so wird das nächste Teil gelegt und geprüft. Dies wiederholt sich solange, bis das neu dazugelegte Teil nicht zu den bereits gelegten Teilen passt. Dann wird dieses Teil entfernt und ein neues gelegt. Sollte keines der aktuell noch verfügbaren Teile passen, so wird das Teil auf dem Index eins vor dem aktuellen entfernt und ein neues gesetzt. Sollte hier auch keines der verfügbaren Teile passen, wird wieder ein Index zurückgegangen. Es wird erst wieder vorangegangen, sobald ein neues gültiges Teil gesetzt werden kann. Gelangt das Programm soweit, dass alle 9 Indices mit gültigen Teilen belegt sind, wurde eine Lösung gefunden. Diese wird abgespeichert und anschließend an den Benutzer ausgegeben. Wurden alle Möglichkeiten durchgegangen und der 8. Index wurde nie mit einem gültigen Teil belegt, so ist das Puzzle unlösbar. Dies wird dann im Anschluss dem Benutzer mitgeteilt.

Das Programm gibt lediglich die erste gefundene Lösung aus. Sobald eine Lösung existiert, gibt es jedoch auf Grund der Drehsymmetrie zwei weitere Lösungen. Es kann aber je nach Puzzle auch noch weitere Lösungen geben.

4. Beispiele

4.1 Beispiel 0

file: examples/puzzel0.txt

Puzzleindex	Piecenumber	Left-Wert	Right-Wert	Center-Wert
0	0	-1	-2	1
1	2	-2	2	-1
2	1	2	-1	-1
3	8	-2	1	-3
4	6	2	2	-1
5	3	-1	3	1
6	4	2	-3	3
7	5	-2	-1	3
8	7	2	-1	-3

4.2 Beispiel 1

file: examples/puzzel1.txt

Puzzleindex	Piecenumber	Left-Wert	Right-Wert	Center-Wert
0	2	-1	-1	3
1	3	-2	1	-1
2	1	2	-3	-1
3	8	2	3	-2
4	7	-2	3	-1
5	6	-2	-3	1
6	5	-1	3	3
7	0	1	-1	2
8	4	3	-1	-3

4.3 Beispiel 2

file: examples/puzzel2.txt

Puzzleindex	Piecenumber	Left-Wert	Right-Wert	Center-Wert
0	0	-2	-1	-3
1	2	-4	1	2
2	8	4	-2	-3
3	4	1	2	3
4	3	-1	1	-3
5	5	-4	-3	-2
6	7	-1	3	4
7	6	1	-2	-3
8	1	1	-2	-4

4.4 Beispiel 3

file: examples/puzzel3.txt

Puzzleindex	Piecenumber	Left-Wert	Right-Wert	Center-Wert
0	0	10	4	10
1	5	3	-4	2
2	8	-3	2	10
3	4	5	-2	6
4	2	-5	10	10
5	3	10	7	-2
6	1	8	-7	9
7	6	-8	10	-6
8	7	10	10	-9

4.5 Auswertung

Es stellt sich heraus, dass alle Puzzlebeispiele gelöst werden können und laufzeittechnisch überhaupt kein Problem darstellen. Die durchschnittliche Laufzeit beträgt 0,062 Milisekunden mit einer Standardabweichung von 0,11 Milisekunden.

5. Denkbare Erweiterungen

Das Programm kann erweitert werden, sodass Dreieckspuzzles beliebiger Größe (4, 9, 16, ...) gelöst werden können. Desweiteren wäre es ein Leichtes das Programm so abzuändern, dass alle existierenden Lösungen ausgegeben werden statt (falls vorhanden) nur die erste gefundene Lösung.

6. Quellcode

(siehe Anhang)

```h
#pragma once
#include <string>
#include <vector>
#include <iostream>
#include <sstream>
#include <fstream>
#include <chrono>  // for high_resolution_clock

#include "piece.h"

using namespace std;

//Variables
auto start_time = std::chrono::high_resolution_clock::now();
auto finish_time = std::chrono::high_resolution_clock::now();
const int number_of_tests = 4;
bool found_solution = false;
vector<bool> Pieces_in_use(9);
vector<vector<piece>> Pieces(9);
vector<piece*> Puzzel(9);
vector<piece*> Solution(9);

//Methods
void print_user_greetings();
void read_input(int number);
void solve(int index);
bool check_new_piece(int index);
void save_solution();
void print_solution();
void cleanup();
int main();
```

```cpp
#include "BwInf39Runde1Aufgabe2.h"


void print_user_greetings()
{
    cout << "Hello there!" << endl;
    cout << "This programm will loop through all the given puzzles and print their solution." << endl;
}

void read_input(int number)
{
    string file_name = "examples/puzzel";
    file_name.append(to_string(number));
    file_name.append(".txt");
    cout << endl << "file: " << file_name << endl;
    ifstream input_file_stream(file_name);

    string dummy;
    getline(input_file_stream, dummy);
    getline(input_file_stream, dummy);

    int left, right, center;
    for (int i = 0; i < 9; i++)
    {
        input_file_stream >> left;
        input_file_stream >> right;
        input_file_stream >> center;

        Pieces[i].push_back(piece(left, right, center, (i * 10 + 0)));
        Pieces[i].push_back(piece(right, center, left, (i * 10 + 1)));
        Pieces[i].push_back(piece(center, left, right, (i * 10 + 2)));
    }
}

void solve(int index)
{
    index++;
    bool valid_new_piece = check_new_piece(index - 1);
    bool reached_end = index == 9;
    if (found_solution)
    {
        return;
    }
    else if (!valid_new_piece)
    {
        return;
    }
    else if (reached_end)
    {
        save_solution();
        found_solution = true;
        return;
    }

    for (int i = 0; i < 9; i++)
```

```
56      {
57          bool piece_in_use = Pieces_in_use[i];
58          if (piece_in_use)
59          {
60              continue;
61          }
62          for (int j = 0; j < 3; j++)
63          {
64              Pieces_in_use[i] = true;
65              Puzzel[index] = &Pieces[i][j];
66              solve(index);
67              Pieces_in_use[i] = false;
68              Puzzel[index] = nullptr;
69          }
70      }
71  }
72
73  bool check_new_piece(int index)
74  {
75      bool match;
76      switch (index)
77      {
78      case -1:
79          return true;
80      case 0:
81          return true;
82      case 1:
83          match = (*Puzzel[0]).right == -(*Puzzel[1]).right;
84          if (match)
85              return true;
86          break;
87      case 2:
88          match = (*Puzzel[1]).left == -(*Puzzel[2]).left;
89          if (match)
90              return true;
91          break;
92      case 3:
93          match = (*Puzzel[2]).right == -(*Puzzel[3]).right;
94          if (match)
95              return true;
96          break;
97      case 4:
98          match = (*Puzzel[3]).left == -(*Puzzel[4]).left;
99          if (match)
100             return true;
101         break;
102     case 5:
103         match = (*Puzzel[1]).center == -(*Puzzel[5]).center;
104         if (match)
105             return true;
106         break;
107     case 6:
108         match = (*Puzzel[5]).right == -(*Puzzel[6]).right;
109         if (match)
110             return true;
111         break;
```

```
112     case 7:
113         match = (*Puzzel[6]).left == -(*Puzzel[7]).left && (*Puzzel[3]).center
               == -(*Puzzel[7]).center;
114         if (match)
115             return true;
116         break;
117     case 8:
118         match = (*Puzzel[6]).center == -(*Puzzel[8]).center;
119         if (match)
120             return true;
121         break;
122     default:
123         return false;
124     }
125     return false;
126 }
127
128 void save_solution()
129 {
130     for (int i = 0; i < 9; i++)
131     {
132         Solution[i] = Puzzel[i];
133     }
134 }
135
136 void print_solution()
137 {
138     bool no_solution_found = Solution[0] == nullptr;
139     if (no_solution_found)
140     {
141         cout << "No solution has been found";
142         return;
143     }
144
145     cout << "Index | Piece number | left, right, and center values" << endl;
146     for (int i = 0; i < 9; i++)
147     {
148         cout << to_string(i) << " -> " << to_string((*Solution[i]).id / 10) <<
               ": " << to_string((*Solution[i]).left) << " " << to_string
             ((*Solution[i]).right) << " " << to_string((*Solution[i]).center) <<
              endl;
149     }
150 }
151
152 void cleanup()
153 {
154     found_solution = false;
155
156     for (int i = 0; i < 9; i++)
157     {
158         Pieces_in_use[i] = false;
159         Puzzel[i] = nullptr;
160         Solution[i] = nullptr;
161
162         Pieces[i].clear();
163     }
```

```
164 }
165
166 int main()
167 {
168     print_user_greetings();
169
170     for (int i = 0; i < number_of_tests; i++)
171     {
172         read_input(i);
173         solve(-1);
174         print_solution();
175         cleanup();
176     }
177
178     string dummy;
179     cin >> dummy;
180
181     return 0;
182 }
183
184
```