Aufgabe 3: Sudokopie

Team-ID: 00246

Team: http://localhost:80/

Bearbeiter/-innen dieser Aufgabe:
Alessio Caputo

13. November 2022

Inhaltsverzeichnis

Lösungsidee

Eine sicherlich gute lösung wäre es durch das Analysieren der freien Felder des Sodokus eine Boolsche Variable zurückzugeben. Da die Aufgabenstellung jedoch vorschreibt, dass der Konstruktionsweg des Sudokus ermittelt werden soll, werden alle möglichen Mutationen des ersten Sudokus mit dem zweiten Sudoku verglichen. Dabei sind alle Mutationswege mit außnahme der Ziffernmutation ohne weiteres umsetzbar. Diese würde schlicht zu viel Zeit beanspruchen. Stattdessen wird jede Position in dem ersten Sudoku mit derselben Position in dem ersten Sudoku verglichen. Dabei wird diese Referenz abgespeichert und für alle weiteren Positionen mit der selben Ziffer überprüft. Die reinfolge in der die Mutationen durchgeführt werden spielen ebenfalls keine Rolle, da prinzipiel ein Vector mit einem anderen Vector addiert wird. So kann aufgrund des Assoziativgesetzes jede mögliche Mutationsreinfolge angenommen werden.

Umsetzung

Sudoku-Klasse

Es bietet sich an, für eine übersichtliche Berechnung eine Klasse Sudoku zu erstellen, die das Sudoku selbst und eine Kommentarvariable enthällt. In der Kommentarvariable können alle Mutationsschritte für eine spätere Ausgabe festgehalten werden. Zusätzlich wird hier die Funktion „isNumberEqual()“ definiert, die überprüft ob ein anderes Sudoku die selbe Struktur mit anderen

Ziffern hat. Dabei werden die Beiden Sudokus mit zwei ineinandergeschachtelten For-Schleifen Feld für Feld miteinander verglichen. Bei dem Vergelichen wird überprüft, ob die Relation der beiden zahlen mit den vorher ermittelten Relationen übereinstimmt. Sollte noch keine Relation zu der aktuellen Ziffer vorhanden sein, wird sie Gespeichert. Sollte es sich um das selbe Sudoku handeln werden alle Relationen zurückgegeben. Sollte dies nicht der Fall sein, wird dies nicht zurückgegeben.

Randomizer-Klasse

Diese Klasse beinhaltet eine Sammlung von Funktionen, von der Jede alle Mutationsmöglichkeiten aus einem Ausgangssudoku berechnet. Dabei Steht eine Funktion für eine Mutationsmöglichkeit. (insgesammt 5) All diese Funktionen Arbeiten mit der vorher Definierten Sudoku Klasse.

90° Rotation im Uhrzeigersinn

In der Aufgabenstellung wird eine 90° Rotation beschriben. Da diese auch mehrfach vorkommen kann, gibt es also 4 mögliche Rotationen (0°, 90°, 180°,270°). Die Funktion muss also aus einem ausgangssudoku 4 weitere Sudokus errechnen, von denen eins das Ausgangssudoku ist. Um die weiteren drei Sudokus zu errechnen wird eine Hilfsfunktion definiert, die ein Sudoku um 90° dreht. Dies wird mit einer For-schleife gelößt, die jede Zeile des Ausgangssudokus in die Spalten des neuen Sudokus übersetzt und anschließend zurückgibt. Das Sudoku wird nun mitt dieser Funktion drei mal Rotiert und die Zwichenergebnisse werden in den Konstruktor, mit einer Beschreibung der Rottion, der Sudoku-Klasse überführt. Anschließend wird eine Liste mit den neuen Rotationen der Sudokuszurückgegeben.

Permutation der drei Spalten & Zeilenblöcke

Um alle Mutationen der Spalten und Zeilenblöcke zu erhalten werden zwei Funktionen definiert (eine für die Spaltenblöcke und eine für die Zeilenblöcke). Wie in der vorher beschriebenen Funktion werden hier alle möglichen Mutationen und eine unveränderte Version des Sudokus zurückgegeben. Zunächst wird eine Liste definiert, die alle Mutationsmöglichkeiten in form von Index-werten festhällt. So wird mit diesen Listen einfach in einer For-schleife das Ausgangssudoku umgeordnet, damit es der Mutation Entspricht. Jedes „neue Sudoku“ wird in einer Liste „variations“ gespeichert, die anschließend zurückgegeben wird.

Permutation der drei Spalten & Zeilen innerhalb der Blöcke

Für diese Mutation wird eine Ähnliche herangehensweise, wie vorher gewählt. Die Mutation der drei Spalten und Zeilenblöcke ist ohne weiteres auf die Spalten und Zeilen innnerhalb der Blöcke übertragbar. Der unterschied besteht darin, dass 3 Zeilenblöcke vorhanden sind. Mit drei ineinandergeschalteten for-schleifen wird für jeden Block das verfahren für die Blöcke angewendet. Die Kommentarvariable wird in jeder For-schleife einzelnt überschrieben, da jeder Zeilenblock seine eigene Mutation hat.

main()

Um alle berechnungen zu machen wird eine Funktion „main()“ definiert, die alle weiteren berechungen und Funktionen des Scriptes ausführt. Zunächst wird die über die Komandozeile weitergegebene Datei, die die Sudokus enthält, eingelesen. Anschließend werden die Sudokus in der Datei seperiert und Sudoku Objekte werden auf deren Basis erstellt.

Mit insgesammt fünf ineinandergeschalteten For-Schleifen werden die ergebnisse der Vorherigen Funktion in die Nächste überführt und anschließend in die Funktion, die die Zahlenasoziazionen überprüft überführt. Sollte deren ergebniss wahr sein findet eine Ausgabe des Konstruktionsweges statt.

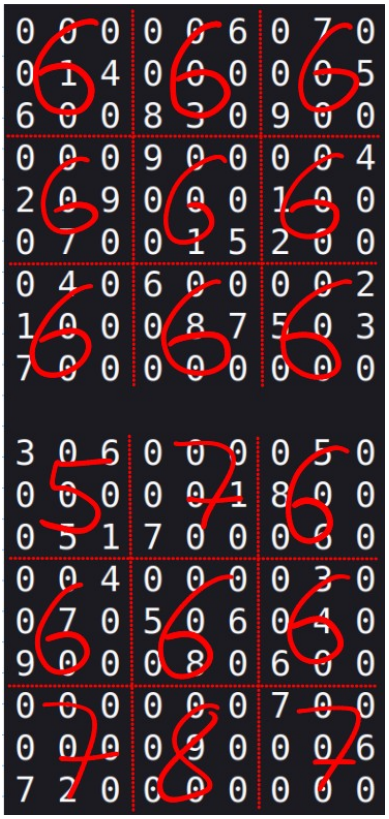
Beispiele

sudoku0.txt	----1---- Mixed in row block 1 from 012 to 102 Mixed in row block 3 from 012 to 021 Mixed in column block 1 from 012 to 201 Mixed in column block 2 from 012 to 120 Mixed in column block 3 from 012 to 210 number relations: 1->1; 2->2; 3->3; 4->4; 5->5; 6->6; 7->7; 8->8; 9->9 ----2---- Mixed in row block 1 from 012 to 201 Mixed in row block 2 from 012 to 210 Mixed in row block 3 from 012 to 120 Mixed in column block 1 from 012 to 102 Mixed in column block 2 from 012 to 021 Mixed column blocks from 123 to 321 Mixed row blocks from 123 to 321 Rotated by 180° number relations: 1->1; 2->2; 3->3; 4->4; 5->5; 6->6; 7->7; 8->8; 9->9
sudoku1.txt	----1---- Mixed column blocks from 123 to 231 Mixed row blocks from 123 to 312 Rotated by 90° number relations: 1->1; 2->2; 3->3; 4->4; 5->5; 6->6; 7->7; 8->8; 9->9 ----2---- Mixed in row block 1 from 012 to 210 Mixed in row block 2 from 012 to 210 Mixed in row block 3 from 012 to 210 Mixed column blocks from 123 to 132 Mixed row blocks from 123 to 213 Rotated by 270° number relations: 1->1; 2->2; 3->3; 4->4; 5->5; 6->6; 7->7; 8->8; 9→9

sudoku2.txt	----1---- Mixed in row block 2 from 012 to 021 Mixed column blocks from 123 to 321 number relations: 1->2; 2->3; 3->4; 4->5; 5->6; 6->7; 7->8; 8->9; 9->1 ----2---- Mixed in row block 1 from 012 to 210 Mixed in row block 2 from 012 to 120 Mixed in row block 3 from 012 to 210 Mixed in column block 1 from 012 to 210 Mixed in column block 2 from 012 to 210 Mixed in column block 3 from 012 to 210 Mixed row blocks from 123 to 321 Rotated by 180° number relations: 1->2; 2->3; 3->4; 4->5; 5->6; 6->7; 7->8; 8->9; 9->1
sudoku3.txt	*keine ausgabe* → keine Lösung
sudoku4.txt	----1---- Mixed in row block 1 from 012 to 021 Mixed in row block 3 from 012 to 021 Mixed column blocks from 123 to 231 Mixed row blocks from 123 to 213 Rotated by 270° number relations: 1->4; 2->8; 3->1; 4->9; 5->2; 6->5; 7->7; 8->3; 9->6 ----2---- Mixed in row block 1 from 012 to 120 Mixed in row block 2 from 012 to 210 Mixed in row block 3 from 012 to 120 Mixed in column block 1 from 012 to 210 Mixed in column block 2 from 012 to 210 Mixed in column block 3 from 012 to 210 Mixed column blocks from 123 to 132 Mixed row blocks from 123 to 312 Rotated by 90° number relations: 1->4; 2->8; 3->1; 4->9; 5->2; 6->5; 7->7; 8->3; 9->6
Zwei identische Sudokus	----1---- number relations: 1->1; 2->2; 3->3; 4->4; 5->5; 6->6; 7->7; 8->8; 9->9 ----2---- Mixed in row block 1 from 012 to 210 Mixed in row block 2 from 012 to 210 Mixed in row block 3 from 012 to 210 Mixed column blocks from 123 to 321 Mixed row blocks from 123 to 321 Rotated by 180° number relations: 1->1; 2->2; 3->3; 4->4; 5->5; 6->6; 7->7; 8->8; 9->9

Auffällig ist, dass sollte es eine Lösung geben, immer zwei Lösungen vorhanden sind. Dies kann mit der Symmetrie des Sudokus erklärt werden. So unterscheidet diese „Parnerkonstruktionswege“ immer ein 180° Winkel. Es kann folglich darauf geschlossen werden, dass eine Drehsymmetrie um 180° vorliegt. Mit diesem Wissen kann gezielt die Hälfte der berechnungen weggelassen werden, da im Falle einer Lösung aus dem Konstruktionsweg auf den anderen Konstruktionsweg geschlossen werden kann. Da nur 2 Lösungen Möglich sind können an diesem Punkt auch alle weiteren Berechnugen vernachlässigt werden. Es sind nicht mehr als 2 Lösungen Möglich, da das Basiskonzept der Sudokus dies nicht zulässt. Für eine Symmetie, die mehr als 2 Lösungen hat müssten 2 Identische Zahlen in der selben Reihe oder Spalte stehen.

Sudoku3.txt hat gezeigt, dass auch keine Lösung möglich ist. Um zu überprüfen, ob das Programm die richtige Lösung errechnet hat, können die Anzahlen der Freien Felder innerhalb der 3x3 Quadrate gezählt werden. Sollten Die Werte nicht alle auch im Zweiten Sudoku vorhanden sein, so kann das Zweite Sudoku nicht aus dem ersten Entstanden sein. (siehe Abbildung) Mit diesem Verfahren kann das Programm Frühzeitig die berechnungen abbrechen und muss so im Falle von keiner Übereinstimmung nicht alle möglichen Mutationen berechnen. Jedoch achtet dieses Verfahren nur auf die freien felder. So ist es auch möglich durch die Zahlenanordnungen keine Lösung zu erhalten.



Quellcode

```
#Alle möglichen Mutationen von sudoku1 mit sudoku2 vergleichen und bei Übereinstimmung ausgeben
variants:int=1
for a in (ramdomizer.getAllMixedInRowBlocksVariants(sudoku1)):
    for b in (ramdomizer.getAllMixedInColumnBlocksVariants(a)):
        for c in (ramdomizer.getAllMixedColumnBlocks(b)):
            for d in (ramdomizer.getAllMixedRowBlocks(c)):
                for e in (ramdomizer.getAllRotations(d)):
                    #überprüfung der Zahlen im Sudoku
                    num = e.isNumberEqualTo(sudoku2.board)
                    if (num["bool"]):
                        print("----"+str(variants)+"----", end="")
                        print(e.comment)
                        print("number relations: "+str(num["numbers"]))
                        variants+=1
```

```
def getAllMixedInRowBlocksVariants(self, sudoku:Sudoku) -> list[Sudoku]:
    s = sudoku.board

    Pvariantions = ["012", "021", "102", "120", "201", "210"]
    variants = []

    for _1 in Pvariantions:
        comment = sudoku.comment
        _one = []
        for a in range(3):
            _one.append(s[int(_1[a])])

        if not _1 == "012":
            commentA = comment + "\nMixed in row block 1 from 012 to "+_1
        else: commentA = comment
        for _2 in Pvariantions:
            _two = _one.copy()
            for b in range(3):
                _two.append(s[int(_2[b])+3])
            if not _2 == "012":
                commentB = commentA + "\nMixed in row block 2 from 012 to "+_2
            else: commentB = commentA
            for _3 in Pvariantions:
                _tree = _two.copy()
                for c in range(3):
                    _tree.append(s[int(_3[c])+6])

                if not _3 == "012":
                    commentC = commentB + "\nMixed in row block 3 from 012 to "+_3
                else: commentC = commentB

                variants.append(Sudoku(_tree, commentC))
    return variants
```

```python
def getAllMixedRowBlocks(self, sudoku:Sudoku) -> list[Sudoku]:
    s = sudoku.board
    Pvariations = [
        {"variation":[0, 1, 2, 3, 4, 5, 6, 7, 8], "comment": sudoku.comment},
        {"variation":[0, 1, 2, 6, 7, 8, 3, 4, 5], "comment": sudoku.comment + "\nMixed row blocks from 123 to 132"},
        {"variation":[3, 4, 5, 0, 1, 2, 6, 7, 8], "comment": sudoku.comment + "\nMixed row blocks from 123 to 213"},
        {"variation":[3, 4, 5, 6, 7, 8, 0, 1, 2], "comment": sudoku.comment + "\nMixed row blocks from 123 to 231"},
        {"variation":[6, 7, 8, 0, 1, 2, 3, 4, 5], "comment": sudoku.comment + "\nMixed row blocks from 123 to 312"},
        {"variation":[6, 7, 8, 3, 4, 5, 0, 1, 2], "comment": sudoku.comment + "\nMixed row blocks from 123 to 321"},
    ]
    variations = []

    for Pvariation in Pvariations:
        v = Pvariation["variation"]
        variation = []
        for i in range(0, 9):
            variation.append([s[v[i]][0], s[v[i]][1], s[v[i]][2], s[v[i]][3], s[v[i]][4], s[v[i]][5], s[v[i]][6], s[v[i]][7], s[v[i]][8]])
        variations.append(Sudoku(variation, Pvariation["comment"]))
    return variations
```

```python
def isNumberEqualTo(self, sudokuboard) -> bool:
    """
    Überprüft, ob die angegebene Sudoku.board Variable schematisch mit self.board übereinstimmt.
    Sollte dies der Fall sein wird ein formatierter String mit den "Nummerrelationen" zurückgegeben.
    """
    #objekt zum Speichen von Relationen
    keys = {"0": "0"}

    #Überprüfung von jedem Zahlenwert in self.board mit sudokuboard
    for x in range(0, 9):
        for y in range(0, 9):
            if self.board[x][y] in keys:
                if not keys[self.board[x][y]] == sudokuboard[x][y]:
                    return {
                        "bool": False
                    }
            else:
                keys[self.board[x][y]] = sudokuboard[x][y]

    #formatieren von keys für Ausgabe
    numberstr:str=""
    for i in range(1, 10):
        numberstr = numberstr+f"{str(i)}->{keys[str(i)]}; "

    return {
        "bool": True,
        "numbers": numberstr[:-2]
    }
```