Duale Hochschule Baden-Württemberg Mannheim

Seminararbeit

Analyse der Lösbarkeit von Instanzen des 15-Puzzles mit Implementierung

Studiengang Informatik

Studienrichtung angewandte Informatik

Verfasser(in):	Kai Fischer, Max Stubenbord
Matrikelnummer:	3683691, 5379506
Kurs:	TINF18AI1
Studiengangsleiter:	Prof. Dr. Holger Hofmann
Wissenschaftliche(r) Betreuer(in):	Prof. Dr. Karl Stroetmann
Bearbeitungszeitraum:	24.03.2021 – 11.06.2021

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Titel "*Analyse der Lösbarkeit von Instanzen des 15-Puzzles mit Implementierung*" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Kai Fischer, Max Stubenbord

Danksagung

Hier können Sie eine Danksagung schreiben.

Kurzfassung (Abstract)

Hier können Sie die Kurzfassung (engl. Abstract) der Arbeit schreiben. Beachten Sie dabei die Hinweise zum Verfassen der Kurzfassung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

AD	Archiv für Diplomatik, Schriftgeschichte, Siegel- und Wappenkunde
BMBF	Bundesministerium für Bildung und Forschung
DHBW	Duale Hochschule Baden-Württemberg
ECU	European Currency Unit
EU	Europäische Union
RDBMS	Relational Database Management System

1 Einleitung

1.1 Geschichte

Die erste bekannte Erscheinung des heute so bekannten Puzzels ist in den 1870'er Jahren in Amerika aufgetreten. Bisher ist man davon ausgegangen, dass der Erfinder des Puzzels der Amerikaner Sam Loyd ist, jedoch ist nach einer Untersuchung von Jerry Slocum and Dieter Gebhardt Sam Loyd gar nicht der echte Erfinder des 15-Puzzels [4, 9]. Demnach hat Sam Loyd die Idee nur gut vermarktet und sich selbst dadurch in die Öffentlichkeit gestellt [1].
`zitat? -> woher die information` Nichtsdestotrotz hat Sam Loyd es geschafft die Aufmerksamkeit der Öffentlichkeit auf das Puzzel zu lenken und somit das Interesse vieler geweckt.
Das Ziel des Puzzels ist es 15 Puzzelsteine numeriert von 1-15 auf einer 4 x 4 Ebene durch Verschiebung in seine Ursprüngliche sortierte Form zurückzubringen (vgl. Abb.1.1).

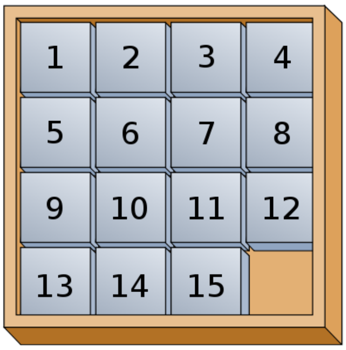

Abbildung 1.1: Zielzustand des 15 Puzzels

Der schnelle Ruhm des Puzzels ergab sich nun daraus, dass Sam Loyd ein Preisgeld von $\$1000$ für denjenigen ausschrieb, der sein Puzzel lösen kann. Darauf folgte ein öffentlicher Ansturm auf das Puzzel, welches als „14-15 Puzzel" bekannt wurde, da lediglich die 14 und 15 vertauscht wurden. (Vgl. Abb.1.2)

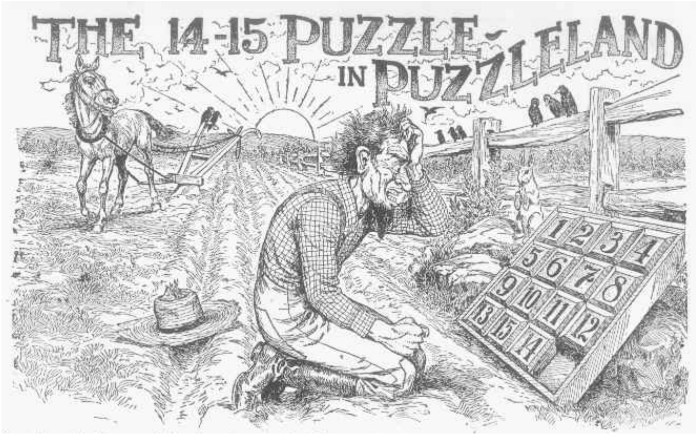

Abbildung 1.2: Illustration des 14-15 Puzzels von Sam Loyd (1914)
Quelle: [8, pp. 234-235]

Allerdings zeigte sich im Nachhinein, dass das Puzzel unlösbar ist, da es eine Transformation von einer geraden zu einer ungeraden Permutation erfordert [1]. Die Unlösbarkeit des Puzzels wurde erstmals von Wm. Woolsey Johnson und William E. Story im Jahre 1879 bei einer Veröffentlichung des American Journal of Mathematics bewiesen. [6] Somit gewann keiner das Preisgeld von $\$1000$. Das 15-Puzzel wurde daraufhin auch bei einer Illustration mit dem Titel „The Great Presidential Puzzle“ der US-Präsidentschaftswahl 1880 referenziert. Zu finden ist dies in der United States Library of Congress's Prints and Photographs division unter der Digitalen ID ppmsca. 15782 [2].

1.2 Aufgabenstellung und Abgrenzung

Hätte man damals bewiesen, dass das Puzzel aus der Einleitung von Sam Loyd nicht lösbar ist, wäre wohl vielen Menschen nächtelange Verzweiflung erspart geblieben.
Nun stellt sich Frage wann eine Instanz des Puzzels lösbar ist. Allgemein gesprochen ist eine Instanz eines 15 Puzzels dann lösbar, falls es eine Sequenz von zulässigen Zügen gibt, welche vom Startzustand zum Zielzustand führen.
Diese Annahme gilt als Zutreffend für genau die hälfte aller möglichen $16! \approx 2 \cdot 10^{13}$ Puzzel Kombinationen [5, 3]. Das Puzzel Problem ist ein klassischer Anwendungsfall wenn es um Modellierung von Algorithmen mit Heuristiken geht. Üblicherweise werden Algorithmen wie *A** oder *IDA** zur Lösung solcher Heurisiken genutzt [1, 3, 7]. Im Rahmen dieser Arbeit geht es nun darum bei gegebener Puzzelinstanz vorherzusagen, ob diese als lösbar oder

unlösbar gilt. Was bei dieser Arbeit nicht betrachtet wird, sind die verschiedenen Algorithmen wie *A** oder auch *IDA**. Diese werden ausfürhlich im zugehörigen Vorlesungsskript in den Sektionen ***2.7*** *A* Search* und ***2.10*** *A*-IDA* Search* erläutert. Lediglich werden diese genutzt um die als lösbar identifizierten Puzzels anschließend auch zu lösen.

1.3 Vorgehen

Das Vorgehen der Arbeit ist dabei wiefolgt:

2 Grundlagen

2.1 Permutationen

2.2 Kontext Vorlesung + Abgrenzung

2.3 Sortieralgorithmen

Mit Ordnung?!

3 Implementation

3.1 Umsetzung

`Der vollständige Quellcode ist im Anhang unter A.1 zu finden.`

Zur Umsetzung wird zu erst eine Datenstruktur definiert, in welcher die Instanzen des 15-Puzzels ausgewertet werden. Hierfür werden Tupel von Tupeln verwendet. Anschließend ist die gernerelle Idee, wie auch schon in `kais abschnitt?` vorgestellt zu schauen, ob die Anzahl der Permutationen bei der Datenstruktur als 1-Dimensionale Liste und der Abstand des Blank-Feldes vom Start- zum Zielzustand die gleiche Parität besitzen. Ist diese gleich, so ist das Puzzel lösbar, vice versa unlösbar.
Spannend ist hier allerdings, dass dies kein Indikator für die Komplexität der Lösbarkeit gibt. So kann ein lösbares Puzzel mit den zur Verfügung stehenden Algorithmen wie *A** oder *IDA** nicht in angemessener Zeit gelöst werden.
Das weitere Vorgehen der Implementierung ist weitesgehend die Bearbeitung von kleinen Teilproblemen, welche sich aus dem eben genannten Vorgehen ergeben.
So betrachten wir zunächst die Anzahl der Permutationen, welche sich in einer Puzzelinstanz befinden. Um diese zu berechnen wird als Datenstruktur eine Liste mit der Dimension 1 benötigt. Anschließend muss die Liste durch tauschen der Elemente sortiert werden. Praktischerweise ist der Lösungszustand des Puzzels so definiert, dass der Index innerhalb der Liste und der Wert des zugehörigen Elementes identisch sind. Somit kann für jeden Index der entsprechende Wert innerhalb der Liste gesucht werden, sodass eine minimale Anzahl von *swaps* durchgeführt werden.
Die Anzahl der getätigten *swaps* liefert uns dann die Anzahl der vorhandenen Permutationen innerhalb der Puzzelinstanz.
Die Implementierung ist dabei wie folgt:

```python
def get_inversion_count(Puzzle: tuple) -> int:
    working_1d_puzzle = to_1d(Puzzle)
    count = 0
    for i in range(len(working_1d_puzzle)):
        if working_1d_puzzle[i] != i:
```

```python
            count += 1
            swap(i, find_tile_1d(i, working_1d_puzzle), working_1d_puzzle)
    return count
```

wobei die Funktion **to_1d** definiert ist durch

```python
to_1d = lambda Puzzle: [elem for tupl in Puzzle for elem in tupl]
```

und die Funktion **swap** durch

```python
def swap(idxA, idxB, Puzzle_1d):
    Puzzle_1d[idxA], Puzzle_1d[idxB] = Puzzle_1d[idxB], Puzzle_1d[idxA]
```

definiert ist. Die Funktion **find_tile_1d** gibt den Index einer 1-Dimensionalen Liste zurück, an der das entsprechende Element zu finden ist. Diese ist auch im Anhanhg unter A.1 zu finden.
Nun muss als nächstes die Distanz des Blank-Feldes vom Startzustand zum Zielszustand berechnet werden. Hierbei wird die x, y Position des Blank-Feldes im Startzustand gesucht und anschließend ähnlich wie bei der Manhattan Distanz die Absolute Differenz beider Zustände addiert. Da die x, y Position des Blank-Feldes im Endzustand immer $0, 0$ ist, muss diese nicht berechnet werden, wird in der Funktion allerdings auch gesucht, sodass bei einer Erweiterung noch der Endzustand variieren kann. Sei α, β nun die Position des Blank-Feldes der Matrix A und γ, ϵ die Position des Blank-Feldes der Matrix B, so ist der Abstand beider Blank-Felder durch

$$|\alpha - \gamma| + |\beta - \epsilon|$$

definiert. Die dazugehörige Funktion **manhattan** ist auch im Anhang unter A.1 zu finden. Zuallerletzt muss nur noch geschaut werden, ob das Produkt der Distanz und die Anzahl der Permutationen im Startzustand die gleiche Parität besitzen. Damit wissen wir nun, ob diese Instanz lösbar ist.

3.2 Testing

Um nun verschiedene Puzzelinstanzen und deren lösbarkeit anhand des Codes zu testen, werden verschiedene Startzustände und zu Testzwecken verschiedene Endzustände definiert. Anschließend werden alle Startzustände zunächst auf lösbarkeit des „normalen“ Endzustandes getestet, danach auf lösbarkeit von allen anderen definierten Endzuständen.
Um eine verbose Ausgabe zu ermöglichen, bei der noch die verschiedenen zuvor berechneten Werte ausgegeben werden können, kann der Funktion, welche die Puzzelinstanzen auf ihre Lösbarkeit prüft noch ein entsprechender verbose Flag mitgegeben werden. Somit kann anhand schon bekannten lösbaren Instanzen überprüft werden, ob die Implementierung vollständig ist.
Der Output dieses Testes befindet sich im dazugehörigen Jupyter-Notebook.
Diese Puzzel werden dann bspw. in das schon aus der Vorlesung bekannte Jupyter-Notebook zum lösen der Instanzen gegeben, sodass geschaut werden kann, ob diese in angemessener Zeit lösbar sind und wieviele Züge für die jeweiliege Lösung gebraucht werden.

4 Zusammenfassung

Literaturverzeichnis

[1] *15 puzzle - Wikipedia*. https://en.wikipedia.org/wiki15_puzzle. (Accessed on 05/30/2021).

[2] *15–14–13. The great presidential puzzle*. http://loc.gov/pictures/resource/ppmsca.15782/. (Accessed on 05/30/2021).

[3] B. Bischoff et al. *Solving the 15-Puzzle Game Using Local Value-Iteration*. 2013. URL: https://mediatum.ub.tum.de/doc/1283911/1283911.pdf.

[4] Jerry Slocum & Dieter Gebhardt. *THE ANCHOR PUZZLE BOOK*. Art of Play, 2012.

[5] Edward Hordern. *Sliding Piece Puzzles*. New York: Oxford University Press, 1986. ISBN: 978-0-198-53204-0.

[6] Wm. Woolsey Johnson und William E. Story. „Notes on the "15"Puzzle“. In: *American Journal of Mathematics* 2.4 (1879), S. 397–404. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2369492.

[7] Richard E. Korf. „Depth-first iterative-deepening: An optimal admissible tree search“. In: *Artificial Intelligence* 27.1 (1985), S. 97–109. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(85)90084-0. URL: https://www.sciencedirect.com/science/article/pii/0004370285900840.

[8] S. Loyd und S. Sloan. *Sam Loyd's Cyclopedia of 5000 Puzzles Tricks and Conundrums with Answers*. Ishi Press International, 2007. ISBN: 9780923891787. URL: https://books.google.de/books?id=noWnPQAACAAJ.

[9] *Review of The 15 Puzzle*. https://www.cut-the-knot.org/books/Reviews/The15Puzzle.shtml. (Accessed on 05/30/2021).

A Beispiel-Anhang: Testanhang

Python Code zur Validierung der Lösbarkeit von Instanzen des 15-Puzzels

Quelltext A.1: Python Code zur Validierung der Lösbarkeit von Instanzen des 15-Puzzels

```python
#!/usr/bin/env python
# coding: utf-8
start1 = {
    'data': ((13, 2, 10, 3),
             (1, 12, 8, 4),
             (5, 0, 9, 6),
             (15, 14, 11, 7)),
    'name': 'start1'}

start2 = {
    'data': ((0,  1,  2,  3),
             (4,  5,  6,  8),
             (14,  7, 11, 10),
             (9, 15, 12, 13)
             ),
    'name': 'start2'
}

start3 = {
    'data': ((6, 13, 7, 10),
             (8, 9, 11, 0),
             (15, 2, 12, 5),
             (14, 3, 1, 4)),
    'name': 'start3'
}
```

```
start4 = {
    'data': ((3, 9, 1, 15),
             (14, 11, 4, 6),
             (13, 0, 10, 12),
             (2, 7, 8, 5)),
    'name': 'start4'
}

start5 = {
    'data': ((1,  2,  3, 4),
             (5,  6,  7, 8),
             (9, 10, 11, 12),
             (13, 15, 14, 0)),
    'name': 'start5'
}

upperLeft = {
    'data': ((0, 1, 2, 3),
             (4, 5, 6, 7),
             (8, 9, 10, 11),
             (12, 13, 14, 15)),
    'name': 'solved - Blank upper Left'
}

downRight = {
    'data': ((1, 2, 3, 4),
             (5, 6, 7, 8),
             (9, 10, 11, 12),
             (13, 14, 15, 0)),
    'name': 'solved - Blank down Right'
}
upperRight = {
    'data': ((1, 2, 3, 0),
             (4, 5, 6, 7),
             (8, 9, 10, 11),
```

```
             (12, 13, 14, 15)),
    'name': 'solved - Blank upper Right'
}
downLeft = {
    'data': ((1, 2, 3, 4),
             (5, 6, 7, 8),
             (9, 10, 11, 12),
             (0, 13, 14, 15)),
    'name': 'solved - Blank down Left'
}
spirale = {
    'data': ((1, 2, 3, 4),
             (12, 13, 14, 5),
             (11, 0, 15, 6),
             (10, 9, 8, 7)),
    'name': 'spirale Goal'
}

Starts = [start1, start2, start3, start4, start5]
Goals = [upperLeft, upperRight, downLeft, downRight, spirale]


def to_1d(Puzzle: tuple) -> list:
    return [elem for tupl in Puzzle for elem in tupl]


def swap(idxA, idxB, Puzzle_1d):
    Puzzle_1d[idxA], Puzzle_1d[idxB] = Puzzle_1d[idxB],
    ↪   Puzzle_1d[idxA]


def find_tile_1d(tile, State_1d):
    n = len(State_1d)
    for it in range(n):
        if State_1d[it] == tile:
```

```python
            return it


def get_inversion_count(Puzzle: tuple) -> int:
    working_1d_puzzle = to_1d(Puzzle)
    count = 0
    old_count = -1
    while old_count != count:
        old_count = count
        for i in range(len(working_1d_puzzle)):
            if working_1d_puzzle[i] != i:
                count += 1
                swap(i, find_tile_1d(i, working_1d_puzzle), working_1d_puzzle)
    return count


def find_tile(tile, State):
    n = len(State)
    for row in range(n):
        for col in range(n):
            if State[row][col] == tile:
                return row, col


def manhattan(stateA, stateB):
    tile = 0
    result = 0
    rowA, colA = find_tile(tile, stateA)
    rowB, colB = find_tile(tile, stateB)
    result += abs(rowA - rowB) + abs(colA - colB)
    return result


def is_solvable(Start: tuple, verbose: bool = False) -> int:
```

```python
    Destination: tuple = upperLeft
    if verbose:
        print(f"Name: {Start['name']}")
        print(f"Start is: {Start['data']}")
        print(f"Inversion Count: {get_inversion_count(Start['data'])}")
        print(
            f"Manhattan Distance: {manhattan(Start['data'], Destination['data'])}")
    return (get_inversion_count(Start['data']) + manhattan(Start['data'], Destination['data'])) % 2 == 0


for s in Starts:
    print(f"is solvable: {is_solvable(s, verbose=True)}")
    print('\n')

get_inversion_count(start4['data'])
```

B Beispiel-Anhang: Noch ein Testanhang

nochmal: lipsum ...