

Aufgabe 3: Hex-Max

Teilnahme-ID: 63175

Bearbeiter/-in dieser Aufgabe:
Lars Noack

17. April 2022

Inhaltsverzeichnis

1	Lösungsidee	1
1.1	Vereinfachungen und Regeln	2
1.2	Das finden der Einzelnen Umlegungen	2
1.3	Die größte Zahl mit unendlich Umlegungen	2
1.4	Die Zahl verringern bis es passt	3
1.5	Mit den übrigen Zügen die Zahl wenn möglich erhöhen	3
2	Umsetzung	3
2.1	Generelles	3
2.2	R und A einer Zahl	3
2.3	größte Zahl ohne Zuglimit	3
2.4	Die Zahl verringern bis es passt	4
2.5	Die übrigen Züge nutzen	4
3	Laufzeitanalyse	5
3.1	größte Zahl ohne Zuglimit	5
3.2	Die Zahl verringern bis es passt	5
3.3	Die übrigen Züge nutzen	5
3.4	praktische Laufzeit	5
4	Beispiele	5
4.1	Winzig	6
4.2	Immer noch klein	6
4.3	nicht groß	6
4.4	ein bisschen groß	8
4.5	immer noch groß	8
4.6	O.o	8
5	Quellcode	9
5.1	Berechnet die zwischenzüge... der Spagetticode tut mir leid.	13

1 Lösungsidee

Da ich sehr viel Zeit für die Bohnusaufgabe verwendet habe, für die mir die Zeit aufgrund von der Schule ausgieng, ist diese Lösung pottentiell nicht ganz ausgereift.

1.1 Vereinfachungen und Regeln

Um meine Lösung zu verstehen ist es wichtig zuerst ein paar Regeln zu etablieren, die Beschreiben wann eine potentielle Lösung richtig sein kann. Da dies einfacher mit Streichhölzern wie mit Sieben-Segment-Anzeigen ist, gehen wir gedanklich davon aus und etablieren die Regel, dass sich die Länge der gegebenen Zahl nicht ändern darf. Dies ist der Fall, da man nicht einfach neue Anzeigen kaufen kann um die Zahl länger zu machen, und nicht einfach Anzeigen nicht nutzen kann, da die Zahl dann kleiner werden würde als ohne überhaupt einen Zug zu machen.

Jetzt ist es wichtig zu beschreiben, was überhaupt ein Zug ist. Ein Zug ist das verschieben eines Streichholzes. Man kann dies aber auch so formulieren, dass man ein Streichholz weglegt, und es wieder hin legt. Dies klingt wie das gleiche, was es aber nicht ist. Definiert ist nämlich nur dass man das Streichholz wieder zurücklegt, und nicht wann. Das heißt man kann eine Ziffer zu einer anderen umwandeln in dem man 2 Streichhölzer wegnimmt. Dann kann man einfach zu anderen Ziffern gehen und diese auch nach belieben umwandeln. Am Ende muss man nur so viele Weggenommen haben, wie man auch Welche hingelegt hat.

Die Anzahl der Weggenommenen Streichhölzer kürze ich zur vereinfachung mit R^1 ab, die Anzahl der Hinzugefügten mit A^2 . m ist die Maximalzahl an Umlegungen bzw. an Zügen und z ist die Zahl an tatsächlich gebrauchten Zügen.

Wenn man schauen will ob ein Ergebniss überhaupt mit beliebigen Umlegungen erreicht werden kann vergleicht man A und R . Wenn beide gleich sind, kann das Ergebniss erreicht werden, sonst nicht. Will man die Anzahl an tatsächlich verbrauchten Zügen wissen gilt $z = R = A$. Ist $z > m$ kann dieses Ergebniss auch nicht erreicht werden, da zu wenig Züge vorhanden sind.

Zusammengefasst gilt

1. Die Zahl an 'Streichhölzer' die weggenommen wurde ist R .
2. Die Zahl an 'Streichhölzer' die hinzugefügt wurden ist A .
3. Die Anzahl an Zügen z ist $z = \max(A, R)$.
4. Wenn $A \neq R$ dann ist die Lösung nicht möglich.
5. Wenn $z > m$ dann ist die Lösung auch nicht möglich.

1.2 Das finden der Einzelnen Umlegungen

In der Aufgabenstellung steht dies: 'Das Programm soll nach jeder Umlegung den Zwischenstand, also die aktuelle Belegung der Positionen ausgeben'. Ich bekomme aber lediglich die größte³ Zahl ohne die Zwischenschritte, was dem geschuldet ist dass ich die 'Streichhölzer' zwischenspeichere und nutze wenn ich will. Die Zwischenschritte zu berechnen ist aber vergleichsweise einfach machbar. Man vergleicht jedes Segment bei jeweils der Startzahl und dem Ergebnis. Wenn dieses Segment verändert wurde merkt man sich das Segment und ob das Segment an oder aus gemacht wurde. Dann schaut man weiter bis man ein Segment gefunden hat, bei dem es statt an, aus gemacht wurde oder anders herum. Dann kann man von der Startzahl die Erste Umlegung bei diesen beiden Segmenten machen. Dies macht man so oft lang, bis man bei dem Ergebniss angekommen ist⁴.

1.3 Die größte Zahl mit unendlich Umlegungen

Wenn ich die größtmögliche Zahl mit unendlichen Umlegungen finde, muss ich diese weniger verändern. Bei dem 5. Beispiel muss ich z.B statt von 0 auf 1369 Züge von 1425 auf 1369 Züge kommen. Dies ist einfacher. Aber wie bekomme ich die größtmögliche Zahl. Um es kurz zu fassen zähle ich die Segmente die bei der gegebenen Zahlenkombination an sind. Dann setzte ich von der vordersten Stelle zur hintersten auf die größtmögliche Zahl. Ausgeschlossen sind alle Zahlen die es verhindern würden, dass die vorhin gezählte Anzahl an Segmenten an ist. Man kann nämlich pro Ziffer höchstens 7 Segmente an machen⁵. Pro Ziffer muss man aber auch mindestens 2 Segmente an machen⁶.

¹number of removed matches

²number of added matches

³vermutlich größte

⁴man sollte diese Zwischenergebnisse natürlich auch ausgeben

⁵8

⁶1

1.4 Die Zahl verringern bis es passt

Das anpassen der größten Zahl mit unendlich Umlegungen mach ich iterativ. Hier beschreibe ich eine iteration, die ich dann wiederhole bis ich nicht mehr zu viele Umlegungen mache.

Ich gehe hier von der kleinsten Stelle zur größten Stelle, da ich den Wert der Zahl so wenig wie möglich verändern will. Bei jeder Ziffer gehe ich dann jede mögliche neue Ziffer durch i . Also alle Zahlen ab 15 bis 0. i ist die Zahl in die ich die Ziffer ändern könnte. Bei jedem neuen i schaue ich dann, ob sich die Umlegungen verringern würde, und ob sich die Tendenz zu A oder R sich nicht verstärkt⁷. Ist dies der Fall, dan setze ich die Ziffer auf i und beende diese eine Iteration. Wenn die nicht der fall ist, dann gehe zum nächsten i oder ggf. zur nächsten Ziffer.

1.5 Mit den übrigen Zügen die Zahl wenn möglich erhöhen

Das iterative herangehen von oben hat das Problem, dass die Züge meist nicht voll ausgeschöpft werden. Dies ist zwar explizit laut der Aufgabenstellung erlaubt, jedoch gibt es oft Situationen, bei denen man mit der Ausschöpfung aller Züge noch das Ergebnis erhöhen kann. Um das auszuschliessen nutze ich brute force.

Simple Konzept. Ich schaue bei jeder Ziffer, bei den großen startend, ob ich diese erhöhen kann, speicher die Differenz zwischen A und R . Dann geh ich zu anderen ziffern und schaue ob ich diese erhöhen kann, so dass dieselbe Differenz $* -1$ einer der gespeicherten Differenzen entspricht. Wenn ändere ich dies.

Somit habe ich die Lösung.

2 Umsetzung

2.1 Generelles

Um R und A überhaupt bekommen zu können, muss ich zuerst die anordnungen für alle 16 ziffern speichern. Dies mach ich in einem bool-array der länge 7. True heißt, das Segment ist an, False heißt es ist aus. Dieses bool-array speicher ich in einer weiteren liste der Länge 16.

Es gibt $15^2 = 225$ Möglichkeiten den Wert einer Ziffer in einen anderen umzuwandeln. Da ich häufig R und A brauche, welches aus dieser Situation entsteht speicher ich das in einer zweidimensionalen Liste ab. Dann kann ich diese werte so bekommen: `changeHex[from][to][add/remove]`

2.2 R und A einer Zahl

Hier geht es nicht um eine Ziffer wie $0xF$ sondern um eine ganze Zahl wie $0xEE4$. R und A dort rauszufinden ist sehr einfach. Ich gehe mit einem for loop über jede ziffer und summiere A und R aus `changeHex` aufeinander.

2.3 größte Zahl ohne Zuglimit

Hier wird es erst interessant. Als erstes brauch ich die maximale Anzahl `maxSeg` und die minnimalen Anzahl `minSeg` an Segmenten die an sind pro Ziffer. Dies wäre 2 und 7. Dann brauch ich auch noch die Anzahl an Segmenten die an sind in der gesamten gegebenen Zahl. Dies nenne ich `leftovers`⁸. Dann initialisiere ich eine leere Liste `hex` die die neue Zahl speichern wird. Dann starte ich ein for loop der die länge der ursprünglichen hex zahl n läuft. Die Anzahl an noch zu füllenden ziffern ist i . Wenn eine Zahl gefunden worde, die funktioniert, wird diese zu `hex` hinzugefügt und von `leftovers` wird die Anzahl an Segmenten der neuen Zahl abgezogen. Um die Zahl zu finden wird bei 15 gestartet und runtegezählt bis eine Zahl valid ist. Eine Zahl ist valid, wenn `minSeg ≤ leftovers : i ≤ maxSeg`.

```

1: hex = []
2: leftovers = givenSegmentAmmount
3: for leftoverDigits = givenHexLen, ..., 0 do
4:     for i = 15, ..., 0 do
5:         SegmentsPerDigit = (leftovers - Segments[i]) / leftoverDigits
6:         if minSeg ≤ SegmentsPerDigit ≤ maxSeg then
```

⁷Die Tendenz kann durch den iterativen Prozess entstehen

⁸wird gleich klar warum

```

7:         hex.append(i)
8:     end if
9: end for
10: end for

```

2.4 Die Zahl verringern bis es passt

Die in Lösungsidee erwähnte Schleife Realisiere ich mit einer while Schleife mit oben genannten bedingungen. Dann iteriere ich mit einem for loop über die Zahl und breche aus dem loop wenn ich etwas geändert habe. Dann schaue ich mit einer for schleife alle möglichen änderungen an, und ob ich die Änderung nehmen sollte, wenn nicht rufe ich continue auf.

Dies mach ich zuerst daran fest, ob die neue tendenz die alte mehr in eine Richtung drückt⁹. Wenn ja dann kann ich dies nicht nehmen. Dannach mus nur die änderung weniger Züge brauchen als die alte, dann nehme ich diese.

```

1: while currentNeededMoves > maxMoves || adds != removes do
2:     for i = lenHex ... 0 do
3:         changed = False
4:         prevTendence = adds - removes
5:         for toDigit = 15 ... 0 do
6:             newTendence = newAdds - newRemoves
7:             if tendence < 0 and newTendence < tendence then
8:                 continue
9:             else if tendence > 0 and newTendence > tendence then
10:                 continue
11:             end if
12:             if oldMoves > newMoves then
13:                 hex[i] = toDigit
14:                 changed = True
15:                 break
16:             end if
17:         end for
18:         if changed then
19:             break
20:         end if
21:     end for
22: end while

```

2.5 Die übrigen Züge nutzen

Nochmal eine kleine Zusammenfassung was ich will. Ich will die Zahl mit meinen übrigen Umlegungen möglichst stark vergrößern. Deshalb fang ich bei den vorderen Stellen an und arbeite mich nach hinten vor, bis keine Züge mehr übrig sind, oder alle Ziffern angeschaut worden sind.

Bei jeder iteration erstelle ich ein dictionary/hash list in dem die Tendenz von jeder Änderung gespeichert wird.

Wenn alles gespeichert gehe ich mit einem ähnlichen for loop, wie der aller erste ab der jetzigen Stelle jede weitere Stelle durch. Dann lass ich so ein for loop wie bei dem auffüllen des dictionarys laufen, bis ich eine änderung gefunden habe, die die Tendenz hat wie -1 * eine aus dem dictionary. Finde ich keine geh ich zur nächsten Stelle.

```

1: for i = 0 ... lenHex do
2:     digit1 = hex[i]
3:     possibleChange = {}
4:     for possibility = 15 ... digit1 + 1 do
5:         possibilityTendence = possibilityAdd - possibilityRemove
6:         if possibilityTendence not in possibleChange then
7:             possibleChange[possibilityTendence] = possibility
8:         end if
9:     end for

```

⁹ Da ich von groß nach klein Zähle, sind in possibleChange immer die größten Werte

⁹Zeile 7 - 11

```

10:   for  $j = i + 1 \dots \text{lenHex}$  do
11:       foundChange = False
12:       for  $\text{possibility} = 15 \dots 0$  do
13:           possibilityTendence = possibilityAdd - possibilityRemove
14:           if  $\text{possibilityTendence} * -1$  in possibleChange then
15:               foundChange = True
16:               hex[i] = possibleChange[ $\text{possibilityTendence} * -1$ ]
17:               hex[j] = possibility
18:               break
19:           end if
20:       end for
21:       if foundChange then
22:           break
23:       end if
24:   end for
25:   if noMovesLeft then
26:       break
27:   end if
28: end for

```

3 Laufzeitanalyse

n ist die Länge der Zahl.

3.1 größte Zahl ohne Zuglimit

Diese Komplexität ist linear. $O(n)$. Der Grund dafür ist, dass der äußere Loop n mal läuft und in dem Loop meist 1 mal oder manchmal mehr als 1 mal. Das mehr als 1 mal ist aber unabhängig von n .

3.2 Die Zahl verringern bis es passt

Die theoretische Laufzeitanalyse hierfür ist schwer und sinnlos. Erstens ist das nicht von n abhängig sondern eher von m . Dies aber auch nicht, da es eigentlich von $\text{zuegeBeiOhneZuglimit} - m \leq 0$ abhängig ist und auf diese Art wird es immer komplexer. Es ist aber meines Erachtens nicht wichtig, da es zwar sehr nah ans Ziel herankommt, aber erst der nächste Schritt, der in jedem Beispiel länger braucht, das Ziel erreicht wird.

3.3 Die übrigen Züge nutzen

Diese Laufzeit sieht ganz anders aus natürlich gibt es dort immer noch das Zufallselement, wie bei den anderen, aber hier spielt das nur eine kleine Rolle, die vernachlässigbar ist.

Die äußerste Schleife läuft n mal wobei n hier wieder die Länge der Zahl ist. In dieser läuft eine Schleife 16 mal was eine Komplexität von $O(16n)$ was gleichzusetzen mit $O(n)$ ist. In der äußeren Schleife ist noch eine Schleife die $n-i$ läuft. i ist hierbei die jeweilige Iteration. Wenn man das von vorn mit dem Kombiniert und die Formel für die Summe aller Integer¹⁰ hinzuzieht kommt man auf $n^2 - n(n+1)/2$. Dies kann als $\frac{n^2-n}{2}$. Wir können sowohl das $-n$ als auch das $/2$ vernachlässigen, was am Ende die Laufzeit $O(n^2)$ ergibt.

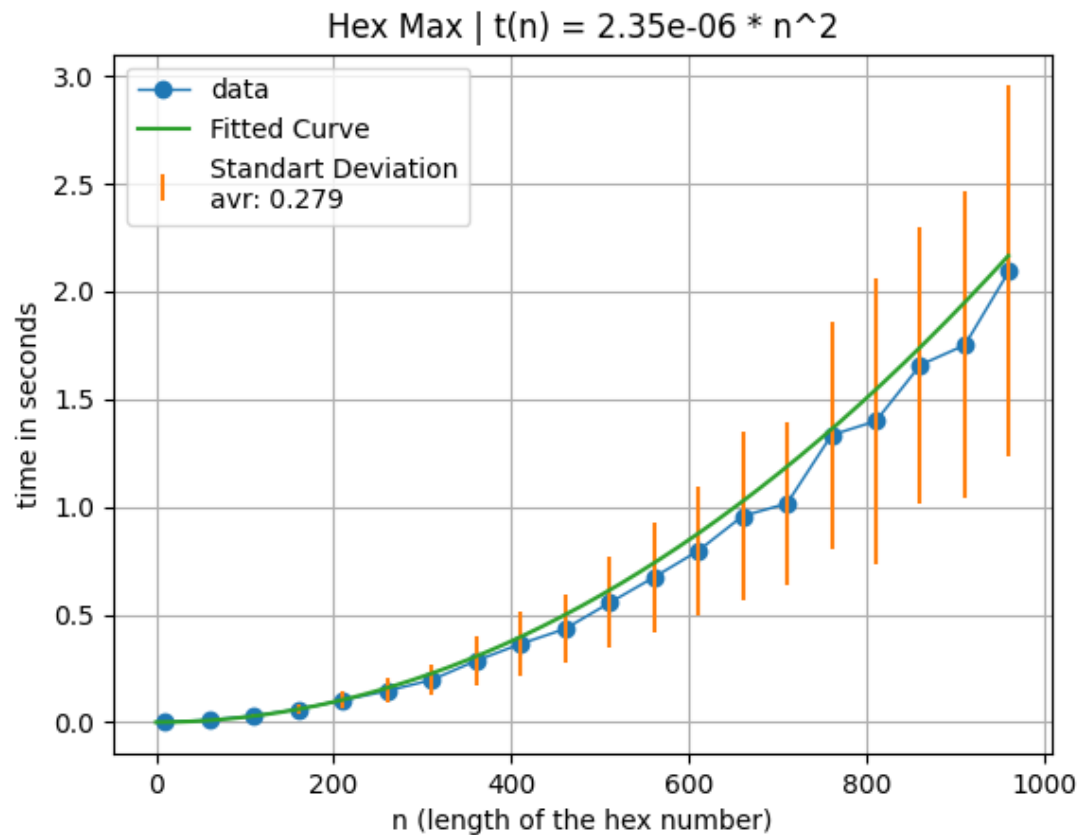
3.4 praktische Laufzeit

In der Untersuchung vernachlässige ich das Berechnen der Zwischenzüge, da dies erstens vernachlässigbar ist. Zweitens da dieser Code eingewoben mit dem Rendering Code ist, welcher schlecht ist und alles langsamer machen kann. Dies ist also schwer zu separieren.

Für die praktische Laufzeitanalyse habe ich für jedes n von 10 bis 1000 in 50er Schritten jeweils 100 Aufgaben generiert und die Zeit gemessen. Daraufhin habe ich für jede 100 Aufgaben den Durchschnitt und die Standardabweichung berechnet. Diese habe ich geplottet, und eine quadratische Funktion an die

¹⁰<https://researchmaniacs.com/Calculator/SumOfIntegers/Sum-of-integers-from-1-to-100.html>

Werte angepasst. Wie man sehen kann passt diese ziemlich gut, was meine Theoretische Laufzeitanalyse von $O(n^2)$ stützt.



4 Beispiele

Ersteinmal vorab, ich habe alle Beispiele gemacht. Um aber alle Zwischenschritte darzustellen, muss ich Bilder rendern, da nicht alle Möglichkeiten einer Sieben-Segment Anzeige einer Zahl entsprechen. Einige Bilder sind jedoch viel zu groß für eine PDF. Diese sind in dem Unterordner 'Solutions' zu finden. Um die Bilder zu öffnen empfehle ich XnView. Alternativ habe ich jeden Schritt, der als Zahl dargestellt werden kann als Zahl im Hexadezimalsystem dargestellt.

m : die maximale Anzahl an umlegungen

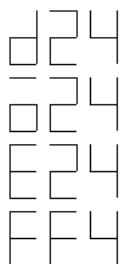
n : die länge der Hex Zahl

4.1 Winzig

$m = 3$

$n = 3$

Startzahl: 0xD24



4.2 Immer noch klein

$$m = 8$$

$$n = 10$$

Startzahl: 0x509C431B55

5	0	9	C	4	3	1	B	5	5
6	0	9	C	4	3	1	B	5	5
F	8	9	C	4	3	1	B	5	5
F	A	8	C	4	3	1	B	5	5
F	A	8	E	4	3	1	B	5	5
F	F	8	E	4	3	1	B	5	5
F	F	A	E	4	3	1	B	5	5
F	F	A	E	4	3	D	B	5	5
F	F	F	E	4	3	D	B	9	5

4.3 nicht groß

Das Bild ist auf der nächsten Seite, da es eine ganze Seite verbraucht. Die nächsten werden nur als text bzw. als screenshot von Text zu sehen sein. Screenshot von Text wegen den wrapping Limitationen von T_EX.

$$m = 37$$

$$n = 40$$

Startzahl: 0x632b29b38f11849015a3bcaee2cda0bd496919f8

8/13

5 Quellcode

Der erste Teil des Codes geht klar.

```

1 import json
2 import numpy as np
3 import logging
4 import enum
5
6 logging.basicConfig(level=logging.DEBUG, format='%(asctime)s_-%(levelname)s_-%(message)s')
7
8 class Hex(enum.Enum):
9     add = "add"
10    remove = "remove"
11    moves = "moves"
12    change = "change"
13
14 # read hex.json and store as hex matrix
15 """
16 hex matrix: list of all states of the seven segment display for every hex number
17 hex sticks: a dictionary with the key being the value of the hex number and the value the number of sticks
18 """
19 with open('hex.json') as f:
20     hex_matrix = json.load(f)
21     hex_sticks = {}
22
23 # count sticks and save as in dictionary
24 for i, number in enumerate(hex_matrix):
25     hex_sticks[i] = np.sum(number)
26     hex_sticks = dict(sorted(hex_sticks.items(), key=lambda x: x[1]))
27
28 # fill up change hex
29 # liste wie viele striche man von jeder zu jeder zahl hinzufuegen bzw. wegnehmen muss.
30 change_hex = {}
31 for hex_1, hex_matrix_1 in enumerate(hex_matrix):
32     change_hex[hex_1] = {}
33
34 for hex_2, hex_matrix_2 in enumerate(hex_matrix):
35     remove = 0
36     add = 0
37     for bool1, bool2 in zip(hex_matrix_1, hex_matrix_2):
38         if bool1 and not bool2:
39             remove += 1
40         elif not bool1 and bool2:
41             add += 1
42
43     change_hex[hex_1][hex_2] = {Hex.add: add, Hex.remove: remove, Hex.moves: max(add, remove), Hex.change: "change"}
44
45 def print_hex_list(hex_list: list):
46     print(''.join([f"{digit:x}" for digit in hex_list]))
47
48 def get_sticks(hex_str: str):
49     sticks = 0
50     for digit in hex_str:
51         sticks += hex_sticks[int(digit, 16)]
52     return sticks
53
54 def get_biggest_hex_infinite(sticks: int, hex_length: int):
55     # get biggest hex number with infinite moves
56     leftovers = sticks
57
58     min_sticks = hex_sticks[list(hex_sticks)[0]]
59     max_sticks = hex_sticks[list(hex_sticks)[-1]]
60
61     # list with integers for each digit
62     hex_number = []
63     for i in range(hex_length):
64         leftover_digits = hex_length - i - 1
65
66         for j in reversed(range(16)):
67             possible_leftovers = leftovers - hex_sticks[j]

```

```

71         if possible_leftovers == 0 and leftover_digits == 0:
72             hex_number.append(j)
73             break
74
75         if leftover_digits == 0:
76             continue
77
78         if possible_leftovers < 0:
79             continue
80
81         sticks_per_digit = possible_leftovers / leftover_digits
82
83         if sticks_per_digit < min_sticks:
84             continue
85         if sticks_per_digit > max_sticks:
86             continue
87
88         leftovers = possible_leftovers
89         hex_number.append(j)
90         break
91
92     if len(hex_number) != hex_length:
93         raise Exception('Numbers dont match while generating the biggest hex number with infinite moves')
94
95     return hex_number
96
97 def needed_value_between_hex(hex_from: list, hex_to: list) -> (int, int):
98     # get needed moves between two hex numbers
99     total_added = 0
100    total_removed = 0
101
102    for from_digit, to_digit in zip(hex_from, hex_to):
103        total_added += change_hex[from_digit][to_digit][Hex.add]
104        total_removed += change_hex[from_digit][to_digit][Hex.remove]
105
106    return total_added, total_removed
107
108 def needed_moves_between_hex(hex_from: list, hex_to: list) -> int:
109     # get needed moves between two hex numbers
110     total_added, total_removed = needed_value_between_hex(hex_from, hex_to)
111
112     if total_added != total_removed:
113         logging.warning(f"{total_added} != {total_removed}")
114         logging.warning(f"{hex_1} -> {hex_2}")
115
116     return max(total_added, total_removed)
117
118 def revert_illegal_moves(hex_from: list, hex_to: list, max_moves: int) -> list:
119     if len(hex_from) <= 0 or len(hex_to) <= 0:
120         raise Exception('hex_from or hex_to is empty. Thus the problem is not in revert_illegal_moves')
121
122     hex_to = hex_to_.copy()
123
124     timeout = 500
125     iteration = 0
126     # the needed_moves in the while loop is performant enough running in about 5 seconds for 5000 iterations
127     needed_moves = max_moves+1
128     needed_change = 0
129     while needed_moves-max_moves > 0 or needed_change != 0:
130         iteration += 1
131         if iteration > timeout:
132             logging.warning('Timeout while reverting moves')
133             return hex_to
134
135         needed_adds, needed_removes = needed_value_between_hex(hex_from, hex_to)
136         # print(needed_adds, needed_removes)
137         needed_moves = max(needed_adds, needed_removes)
138         needed_change = needed_adds - needed_removes
139
140     if not (needed_moves-max_moves > 0 or needed_change != 0):
141         break
142
143

```

```

145     # print_hex_list(hex_to)
146     # print(f"{needed_adds} - {needed_removes} = {needed_change} -> {needed_moves}")
147
148     for i, from_digit in reversed(list(enumerate(hex_to))):
149         if hex_from[i] == from_digit:
150             continue
151
152         initial_adds = change_hex[hex_from[i]][from_digit][Hex.add]
153         initial_removes = change_hex[hex_from[i]][from_digit][Hex.remove]
154         initial_moves = max(initial_adds, initial_removes)
155
156         # die adds und removes die in dieser iteration nicht geaendert werden koennen
157         # also von inf change ohne diese iteration
158         bare_adds = needed_adds - initial_adds
159         bare_removes = needed_removes - initial_removes
160
161         # print(f"{i} {from_digit} move {needed_moves} change {needed_change} specific a{adds} r{removes}")
162         new_digit = -1
163         for to_digit in reversed(range(16)):
164             iter_adds = bare_adds + change_hex[hex_from[i]][to_digit][Hex.add]
165             iter_removes = bare_removes + change_hex[hex_from[i]][to_digit][Hex.remove]
166
167             iter_change = iter_adds - iter_removes
168             iter_moves = max(iter_adds, iter_removes)
169
170             # print(f"iter {i} {to_digit} add {iter_adds} remove {iter_removes} move {iter_moves} change {iter_change}")
171
172             if needed_change > 0 and iter_change > needed_change: continue
173             if needed_change < 0 and iter_change < needed_change: continue
174
175             # print("heh", iter_adds, iter_removes)
176             if iter_moves < needed_moves:
177                 # print("????????????????")
178                 new_digit = to_digit
179                 break
180
181             if new_digit != -1:
182                 break
183
184         if new_digit == -1:
185             pass
186             # print_hex_list(hex_to)
187             # logging.warning('No digit found to revert illegal moves (if it occurs only once, it is not a problem)')
188         else: hex_to[i] = new_digit
189
190     return hex_to
191
192
193 def fill_up_moves(hex_from: list, hex_to: list, max_moves: int) -> list:
194     if len(hex_from) <= 0 or len(hex_to) <= 0:
195         raise Exception('hex_from or hex_to is empty. Thus the problem is not in fill_up_moves')
196     hex_to = hex_to.copy()
197
198     for i, digit1 in enumerate(hex_to):
199         if digit1 == 15:
200             continue
201         hex_change = {}
202
203         initial_change = change_hex[hex_from[i]][digit1][Hex.add] - change_hex[hex_from[i]][digit1][Hex.remove]
204         for possibility in reversed(range(digit1 + 1, 16)):
205             possible_change = change_hex[hex_from[i]][possibility][Hex.add] - change_hex[hex_from[i]][possibility][Hex.remove]
206
207             total_change = initial_change - possible_change
208             if total_change not in hex_change:
209                 hex_change[total_change] = []
210             hex_change[total_change].append(possibility)
211         found_anything = False
212
213     for j, digit2 in reversed(list(enumerate(hex_to))[i+1:]):
214         initial_change = change_hex[hex_from[i]][digit2][Hex.add] - change_hex[hex_from[i]][digit2][Hex.remove]

```

```

217         for possibility in reversed(range(16)):
218             possible_change = change_hex[hex_from[i]][possibility][Hex.add] - change_hex[hex_from[i]
219                 Hex.remove]
220
221             total_change = initial_change - possible_change
222
223             if -total_change in hex_change:
224
225                 prev_i = hex_to[i]
226                 prev_j = hex_to[j]
227
228                 hex_to[i] = max(hex_change[-total_change])
229                 hex_to[j] = possibility
230
231                 if needed_moves_between_hex(hex_from, hex_to) > max_moves:
232                     hex_to[i] = prev_i
233                     hex_to[j] = prev_j
234
235             else:
236                 found_anything = True
237                 break
238
239         if found_anything:
240             break
241         if needed_moves_between_hex(hex_from, hex_to) >= max_moves:
242             break
243
244
245
246
247     return hex_to
248
249 def execute_file(file_number: int = 0) -> list:
250     print("\n\n#####")
251     # read example file
252     with open(f'examples/hexamax{file_number}.txt') as f:
253         hex_str, moves = f.read().splitlines()
254         moves = int(moves)
255         hex_number = [int(digit, 16) for digit in hex_str]
256         print(f'Moves: {moves}')
257         print(f'Hex: {hex_str}')
258
259     hex_length = len(hex_str)
260     stick_count = get_sticks(hex_str)
261
262     print(f'hex_length: {hex_length}')
263     print(f'Sticks: {stick_count}')
264
265     with_infinite_moves = get_biggest_hex_infinite(stick_count, hex_length)
266     print_hex_list(with_infinite_moves)
267
268     # get needed moves between hex numbers
269     needed_moves = needed_moves_between_hex(hex_number, with_infinite_moves)
270     print(f'Needed moves: {needed_moves}')
271
272     # revert the moves that arent possible
273     if needed_moves <= moves:
274         return with_infinite_moves
275
276     temp_solution = revert_illegal_moves(hex_number, with_infinite_moves, moves)
277     print_hex_list(temp_solution)
278     solution = fill_up_moves(hex_number, temp_solution, moves)
279     print_hex_list(solution)
280     # print(needed_value_between_hex(hex_number, solution))
281     print(needed_value_between_hex(hex_number, solution))
282     return hex_number, moves, solution, needed_value_between_hex(hex_number, solution)
283
284
285
286
287 if __name__ == '__main__':

```

```
sollution, meta = execute_file(5)
```

5.1 Berechnet die zwischenzüge... der Spagetticode tut mir leid.

Die hälfte der Funktionen fehlen hier, da diese nur fürs Zeichnen sind.

```
def draw(hex_number: list, sollution: list, moves: int, example: int):
2     with open(f'solutions/hexmax{example}.txt', "w") as f:
        pass
4
        length = 20
6        gap = 10
        thickness = 1
8        digits = len(hex_number)
        # create pillow image
10       img = Image.new('RGB', ((length + gap) * digits + gap, (length + length + gap) * (moves + 1) + gap))

12       start_hex = [hex_matrix[i].copy() for i in hex_number]
        end_hex = [hex_matrix[i].copy() for i in sollution]
14
        save_in_text(start_hex, example)
16
        img = draw_digits(img, start_hex, 0, moves + 1, length=length, gap=gap, thickness=thickness)
18       for i in range(moves):
            add = False
20            remove = False
            to_break = False
22            for j, (digit1, digit2) in enumerate(zip(start_hex, end_hex)):
                for k, (bit1, bit2) in enumerate(zip(digit1, digit2)):
24                    if bit1 and (not bit2) and not add:
                        add = True
26                    start_hex[j][k] = bit2
                        if remove:
28                            to_break = True
                                    break
30                    elif (not bit1) and bit2 and not remove:
                        remove = True
32                    start_hex[j][k] = bit2
                        if add:
34                            to_break = True
                                    break
36
                if to_break:
38                    break

40            save_in_text(start_hex, example)
            img = draw_digits(img, start_hex, i + 1, moves + 1, length=length, gap=gap, thickness=thickness)
42
        img.save(f'solutions/hexmax{example}.png')
```