Aufgabe 3: Hex-Max

Teilnahme-ID: 63175

Bearbeiter/-in dieser Aufgabe:
Lars Noack

20. April 2022

Inhaltsverzeichnis

1 Lösungsidee

Da ich sehr viel Zeit für die Bonusaufgabe verwendet habe, für die mir die Zeit aufgrund von der Schule ausging, ist diese Lösung potenziell 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 potenzielle Lösung richtig sein kann. Da dies einfacher mit Streichhölzern als mit Sieben-Segment-Anzeigen geht, 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. Auch kann man nicht einfach Anzeigen auslassen, da die Zahl dann kleiner werden würde 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 hinlegt. Das klingt wie das gleiche, was es aber nicht ist. Definiert ist nämlich nur, dass man das Streichholz wieder zurücklegt und nicht wann das passieren muss. Das heißt, man kann eine Ziffer zu einer anderen umwandeln, indem man 2 Streichhölzer wegnimmt. Dann kann man einfach zu anderen Ziffern gehen und auch diese nach Belieben umwandeln. Am Ende muss man nur so viele weggenommen haben, wie man auch 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 erlaubte Maximalzahl an Umlegungen bzw. an Zügen und z ist die Zahl an tatsächlich gebrauchten Zügen.

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

Zusammengefasst gilt

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

[1] number of removed matches
[2] number of addet matches

1.2 Das Finden der einzelnen Umlegungen

In der Aufgabenstellung steht: 'Das Programm soll nach jeder Umlegung den Zwischenstand, also die aktuelle Belegung der Positionen ausgeben'. Ich bekomme aber lediglich die größte[3] Zahl ohne die Zwischenschritte, was der Tatsache 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 der jeweiligen 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, das ebenfallsverändert wurde. Danach kann man von der Startzahl aus die erste Umlegung bei diesen beiden Segmenten machen. Dies macht man so oft lang, bis man bei dem Ergebnis angekommen ist[4].

1.3 Die größte Zahl mit unendlichen Umlegungen

Wenn ich die größtmögliche Zahl mit unendlichen Umlegungen finde, muss ich diese weniger verändern. Bei dem 5. Beispiel muss ich z.B von 0 auf 1369 Züge kommen, statt von 1425 auf 1369 Züge. 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 setze ich von der vordersten Stelle aus zur hintersten auf die größtmöglichte 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 machen[5]. Pro Ziffer muss man aber auch mindestens 2 Segmente an machen[6].

1.4 Die Zahl verringern, bis es passt

Das anpassen der größten Zahl mit unendlichen Umlegungen mache 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 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 nicht verstärkt[7]. Ist dies der Fall, dan setze ich die Ziffer auf i und beende diese eine Iteration. Wenn das 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

Die iterative Herangehensweise von oben hat das Problem, dass die Züge meist nicht voll ausgeschöpft werden. Dies ist zwar explitzit 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.

Simples Konzept. Ich schaue bei jeder Ziffer, anfangend bei den Größten, ob ich diese erhöhen kann. Wenn ja, speichere die Differenzen zwischen A und R. Dann gehe ich zu anderen Ziffern und schaue ob ich diese erhöhen kann, sodass dieselbe Differenz $\cdot -1$ einer der gespeicherten Differenzen entspricht. Wenn ja, 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 mache ich in einem bool-array der Länge 7. True heißt, dass das Segment an ist, False heißt es ist aus. Diesen bool-array speichere ich in einer weiteren Liste der Länge 16.

[3] vermutlich größte
[4] man sollte diese Zwischenergebnisse natürlich auch ausgeben
[5] 8
[6] 1
[7] Die Tendenz kann durch den iterativen Prozess entstehen

Es gibt $15^2 = 225$ Möglichkeiten den Wert einer Ziffer in einen anderen umzuwandeln. Da ich häufig R und A brauche, welche aus dieser Situation entstehen, speichere ich diese in einer zweideminsionalen Liste ab. Dann kann ich die 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 herauszufinden 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 brauche ich die maximale Anzahl $maxSeg$ und die minnimale Anzahl $minSeg$ an Segmenten die an sind, pro Ziffer. Dies wäre 2 und 7. Dann brauche ich auch noch die Anzahl an Segmenten die in der gesammten gegebenen Zahl an sind. Diese nenne ich $leftovers$[8]. Dann initialisiere ich eine leere Liste hex, die die neue Zahl speichern wird. Danach starte ich einen for-loop der die Länge der ursprünglichen hex-Zahl n läuft (pro Ziffer eine Iteration). Die Anzahl an noch zu füllenden Ziffern ist i. Wenn eine Zahl gefunden wurde 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 heruntegezählt, bis eine Zahl valid ist. Eine Zahl ist valid, wenn $minSeg \leq leftovers : i \leq maxSeg$.

```
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
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 den oben genannten Bedingungen. Dann iteriere ich mit einem for-loop über die Zahl und breche den loop, wenn ich etwas geändert habe. Anschließend schaue ich mit einer for-Schleife alle möglichen Änderungen an, und ob ich die Änderung übernehmen sollte. Wenn nicht, rufe ich continue auf.

Dies mach ich zuerst daran fest, ob die neue Tendenz die alte mehr in eine Richtung drückt[9]. Wenn ja, dann kann ich sie nicht nehmen. Dannach muss 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
```

[8] wird gleich klar warum
[9] Zeile 7 - 11

```
            end if
        end for
        if changed then
            break
        end if
    end for
end while
```

2.5 Die übrigen Züge nutzen

Nochmal eine kleine Zusammenfassung von dem, was ich will. Ich will die Zahl mit meinen übrigen Umlegungen möglichst stark vergrößern. Deshalb fange 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 temporäres dictionary, in dem die Tendenz von jeder Änderung gespeichert wird.

Wenn alles gespeichert ist, gehe ich mit einem ähnlichen for-loop wie der aller erste, ab der jetzigen Stelle jede weitere Stelle durch. Dann lasse ich einen ähnlichen for-loop wie bei dem Auffüllen des dictionarys laufen, bis ich eine Änderung gefunden habe, die die Tendenz hat wie eine aus dem dictionary $\cdot -1$. Finde ich keine, gehe ich zur nächsten Stelle.

```
for i = 0...lenHex do
    digit1 = hex[i]
    possibleChange = {}
    for possibility = 15...digit1 + 1 do
        possibilityTendence = possibilityAdd - possibilityRemove
        if possibilityTendence not in possibleChange then
            possibleChange[possibilityTendence] = posibility
        end if   ▷ Da ich von groß nach klein zähle, sind in possibleChange immer die größten Werte
    end for
    for j = i + 1...lenHex do
        foundChange = False
        for possibility = 15...0 do
            possibilityTendence = possibilityAdd - possibilityRemove
            if possibilityTendence * -1 in possibleChange then
                foundChange = True
                hex[i] = possibleChange[possibilityTendence * -1]
                hex[j] = possibility
                break
            end if
        end for
        if foundChange then
            break
        end if
    end for
    if noMovesLeft then
        break
    end if
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 und der innere loop meist ein mal oder manchmal mehr als ein mal läuft. Wenn er mehr als ein mal läuft, ist dies 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 eigentlich auch nicht, da es eigentlich von $genutzteZuegeOhneZuglimit - m \leq 0$ abhängig ist und es auf diese Art immer komplexer wird. Es ist aber meines Erachtens nicht wichtig, da es sehr nah ans Ziel herankommt. Aber erst der nächste Schritt, der in jedem Beispiel länger braucht, erreicht das Ziel.

3.3 Die übrigen Züge nutzen

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

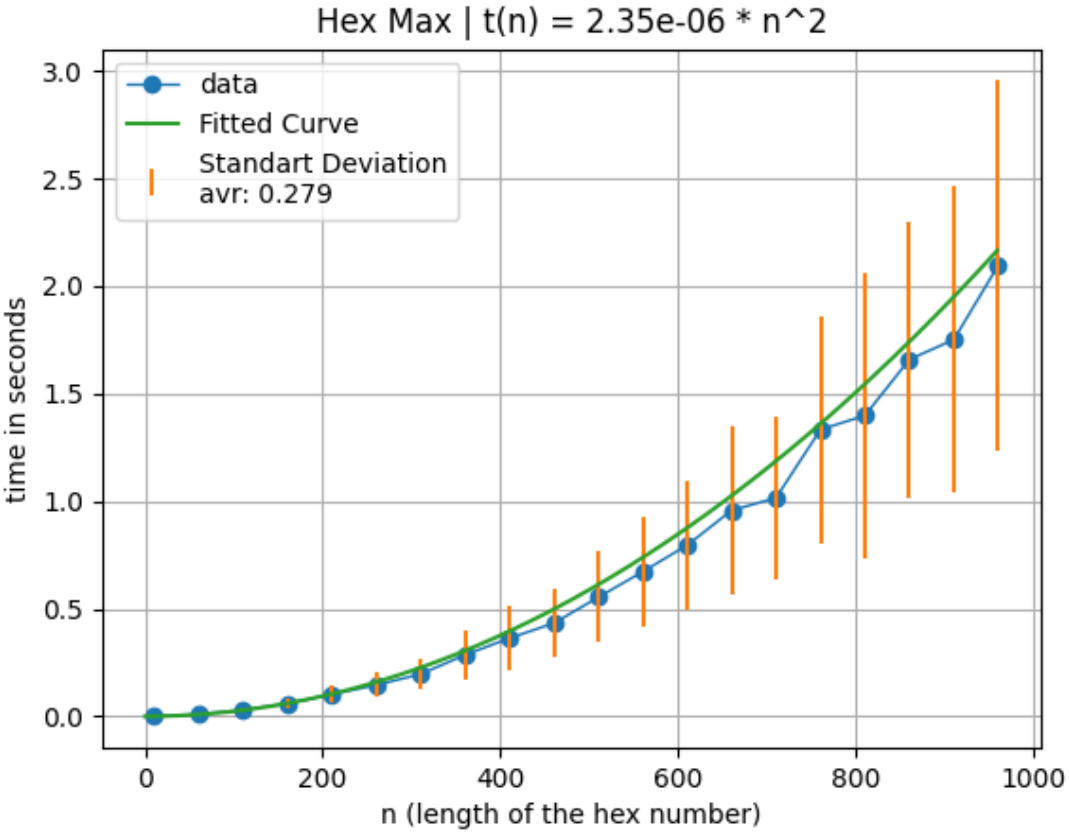
Die äußerste Schleife läuft n mal, wobei n hier wieder die Länge der Zahl ist. In dieser läuft eine Schleife 16 mal, was eine Komplexität von $O(16n)$, gleichzusetzen mit $O(n)$, ist. In der äußeren Schleife ist noch eine Schleife die $n - i$ läuft. i ist hierbei die jeweilige Iteration. Wenn man das von vorhin mit dem kombiniert, und die Formel für die Summe aller Integer [10] 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 vernachlässigbar ist. Auch ist dieser code eingewoben mit dem rendering code, welcher schlecht ist und alles langsamer machen kann. Dies ist also schwer zu seperieren.

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 Messung bei einer bestimmten Größe von n den Durchschnitt und die Standartabweichung berechnet. Diese habe ich geplottet, und eine quadratische Funktion an die Werte angepasst. Wie man sehen kann passt diese zimlich gut, was meine Theoretische Laufzeitanalyse von $O(n^2)$ stützt.

[10] `https://researchmaniacs.com/Calculator/SumOfIntegers/Sum-of-integers-from-1-to-100.html`



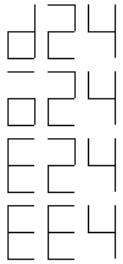
4 Beispiele

Erst einmal 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

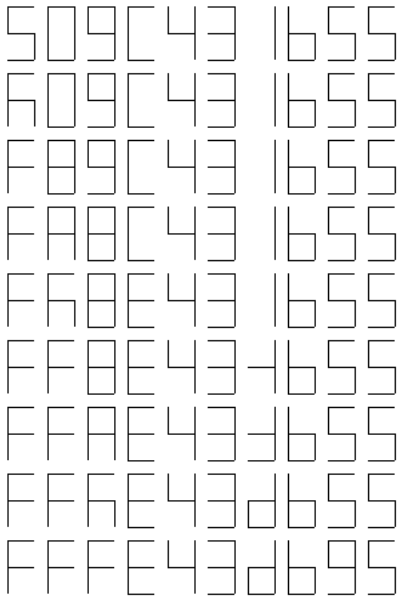
$m = 3$
$n = 3$
Startzahl: 0xD24



4.2 Immer noch klein

$m = 8$
$n = 10$

Startzahl: 0x509C431B55

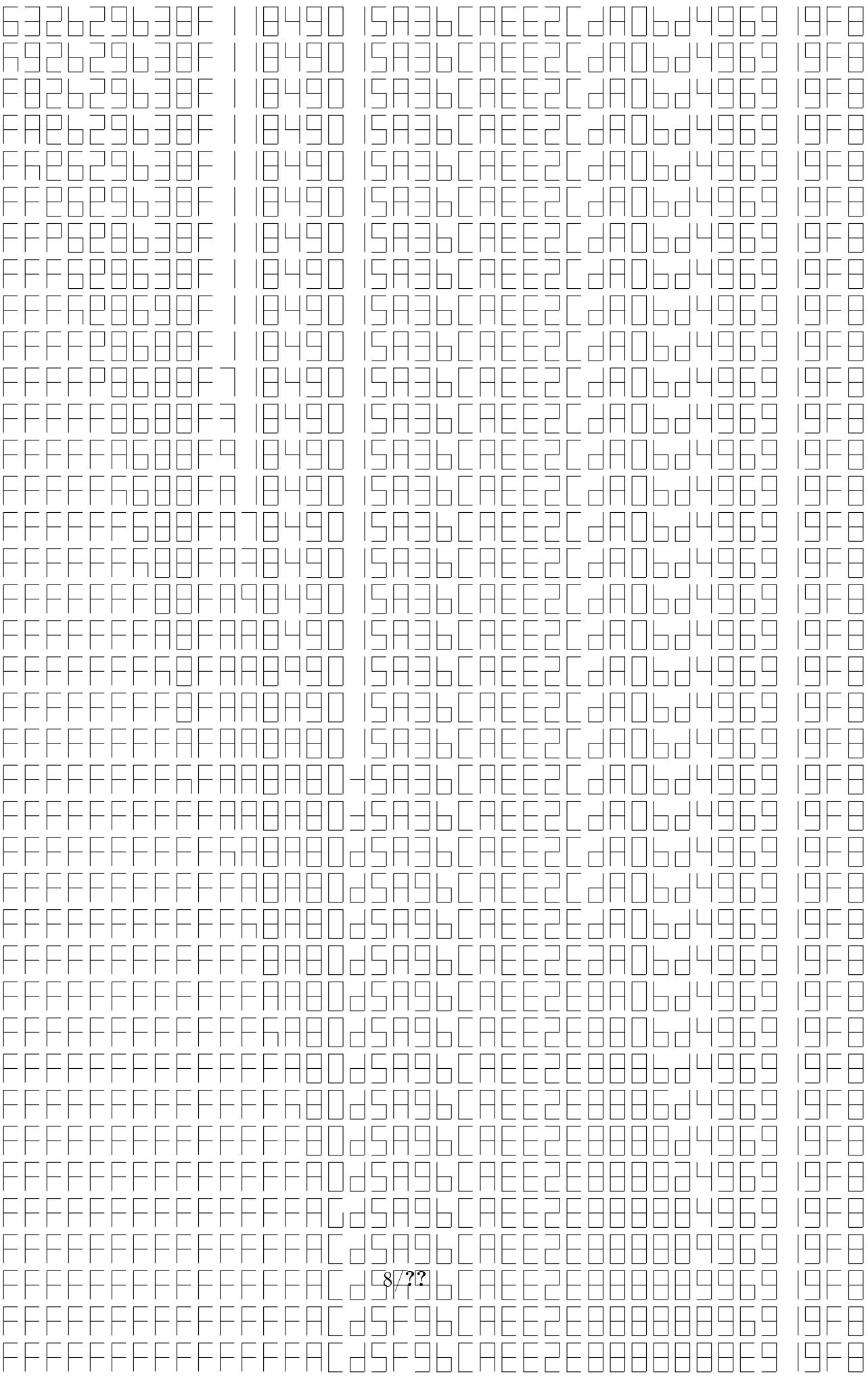


4.3 nicht groß

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 TEX.

$m = 37$
$n = 40$

Startzahl: 0x632b29b38f11849015a3bcaee2cda0bd496919f8


8//??

4.4 ein bischen groß

$m = 121$
$n = 100$

Startzahl:
0x0e9f1db46b1e2c081b059eaf198fd491f477ce1cd37ebfb65f8d765055757c6f4796bb8
b3df7fcac606dd0627d6b48c17c09

größte Zahl:
0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffeff479ebb8
b3df7a888888888888888888a7c09

4.5 immer noch groß

$m = 87$
$n = 100$

Startzahl:
0x1a02b6b50d7489d7708a678593036fa265f2925b21c28b4724dd822038e3b4804192322
f230ab7af7bda0a61ba7d4ad8f888

größte Zahl:
0xffffffffffffffffffffffffffffffffffffae5b2dc2fb4724dda22038e3b4808888888
88888888888da0a61ba7d4ad8f888

4.6 O.o

$m = 1369$
$n = 1000$

Startzahl:
0xef50aa77ecad25f5e11a307b713eaaec55215e7e640fd263fa529bbb48dc8fafe14d5b0
2ebf792b5ccbbe9fa1330b867e330a6412870dd2ba6ed0dbcae553115c9a31ff350c5df99
3824886db5111a83e773f23ad7fa81a845c11e22c4c45005d192ade68aa9aa57406eb0e7c
9ca13ad03888f6abedf1475fe9832c66bfdc28964b7022bdd969e5533ea4f2e4eaba75b5d
c11972824896786bd1e4a7a7748fdf1452a5079e0f9e6005f040594185ea03b5a869b109a
283797ab31394941bfe4d38392ad12186ff6d233585d8c820f197fba9f6f063a0877a912c
cbdcb14beecbaec0ed061cff60bd517b6879b72b9efe977a9d3259632c718fbf45156a165
76aa7f9a4fad40ad8bc87ec569f9c1364a63b1623a5ad559aaf6252052782bf9a46104e44
3a3932d25aae8f8c59f10875fad3cbd885ce68665f2c826b1e1735ee2fdf0a196514 9df35
3ee0be81f3ec133922ef43ebc09ef755fbd740c8e4d024b033f0e8f3449c94102902e1434
33262cda1925a2b7fd01bef26cd51a1fc22edd49623ee9deb14c138a7a6c47b677f033bde
b849738c3ae5935a2f54b99237912f2958fdfb82217c175448aa8230fdcb3b3869824a826
635b538d47d847d8479a88f350e24b31787dfd60de5e260b265829e036be340ffc0d8c055
55e75092226e7d54deb42e1bb2ca9661a882fb718e7aa53f1e606

größte Zahl:
0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffea82ff43ebc09ef755fbd740c8e4d024888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888
88888888888888888888888888888888888888888e7aa53f1e60f

5 Quellcode

Der erste Teil des Codes geht klar. Wollen Sie aber guten code sehen, schauen Sie sich den von Rechenrätsel an. Hier war ich unter Zeitdruck.

```
import json
import numpy as np
import logging
import enum

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

class Hex(enum.Enum):
    add = "add"
    remove = "remove"
    moves = "moves"
    change = "change"

# read hex.json and store as hex matrix
"""
hex matrix: list of all states of the seven segment display for every hex number
hex sticks: a dictionary with the key being the value of the hex number and the value the number of sti
"""
with open('hex.json') as f:
    hex_matrix = json.load(f)
    hex_sticks = {}

    # count sticks and save as in dictionary
    for i, number in enumerate(hex_matrix):
        hex_sticks[i] = np.sum(number)
    hex_sticks = dict(sorted(hex_sticks.items(), key=lambda x: x[1]))


# fill up change hex
# liste wie viele striche man von jeder zu jeder zahl hinzufuegen bzw. wegnehmen muss.
change_hex = {}
for hex_1, hex_matrix_1 in enumerate(hex_matrix):
    change_hex[hex_1] = {}

    for hex_2, hex_matrix_2 in enumerate(hex_matrix):
        remove = 0
        add = 0
        for bool1, bool2 in zip(hex_matrix_1, hex_matrix_2):
            if bool1 and not bool2:
                remove += 1
            elif not bool1 and bool2:
                add += 1

        change_hex[hex_1][hex_2] = {Hex.add: add, Hex.remove: remove, Hex.moves: max(add, remove), Hex.


def print_hex_list(hex_list: list):
    print(''.join([f"{digit:x}" for digit in hex_list]))

def get_sticks(hex_str: str):
    sticks = 0
    for digit in hex_str:
        sticks += hex_sticks[int(digit, 16)]
    return sticks


def get_biggest_hex_infinite(sticks: int, hex_length: int):
    # get biggest hex number with infinite moves
    leftovers = sticks

    min_sticks = hex_sticks[list(hex_sticks)[0]]
    max_sticks = hex_sticks[list(hex_sticks)[-1]]

    # list with integers for each digit
    hex_number = []
    for i in range(hex_length):
        leftover_digits = hex_length - i - 1
```

```
69          for j in reversed(range(16)):
                possible_leftovers = leftovers - hex_sticks[j]
71
                if possible_leftovers == 0 and leftover_digits == 0:
73                  hex_number.append(j)
                    break
75
                if leftover_digits == 0:
77                  continue

79              if possible_leftovers < 0:
                    continue
81              sticks_per_digit = possible_leftovers / leftover_digits

83              if sticks_per_digit < min_sticks:
                    continue
85              if sticks_per_digit > max_sticks:
                    continue
87
                leftovers = possible_leftovers
89              hex_number.append(j)
                break
91
    if len(hex_number) != hex_length:
93          raise Exception('Numbers dont match while generating the biggest hex number with infinite moves

95      return hex_number

97  def needed_value_between_hex(hex_from: list, hex_to: list) -> (int, int):
        # get needed moves between two hex numbers
99      total_added = 0
        total_removed = 0
101
        for from_digit, to_digit in zip(hex_from, hex_to):
103         total_added += change_hex[from_digit][to_digit][Hex.add]
            total_removed += change_hex[from_digit][to_digit][Hex.remove]
105
        return total_added, total_removed
107
    def needed_moves_between_hex(hex_from: list, hex_to: list) -> int:
109     # get needed moves between two hex numbers
        total_added, total_removed = needed_value_between_hex(hex_from, hex_to)
111
        if total_added != total_removed:
113         logging.warning(f"{total_added} != {total_removed}")
            logging.warning(f"{hex_1} -> {hex_2}")
115
        return max(total_added, total_removed)
117
    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:
            raise Exception('hex_from or hex_to is empty. Thus the problem is not in revert_illegal_moves')
121
        hex_to = hex_to_.copy()
123
        timeout = 500
125     iteration = 0
        # the needed_moves in the while loop is performant enough running in about 5 seconds for 5000 itera
127     needed_moves = max_moves+1
        needed_change = 0
129     while needed_moves-max_moves > 0 or needed_change != 0:
            iteration += 1
131         if iteration > timeout:
                logging.warning('Timeout while reverting moves')
133             return hex_to


135
            needed_adds, needed_removes = needed_value_between_hex(hex_from, hex_to)
137         # print(needed_adds, needed_removes)
            needed_moves = max(needed_adds, needed_removes)
139         needed_change = needed_adds - needed_removes

141         if not (needed_moves-max_moves > 0 or needed_change != 0):
```

```
            break

        # print_hex_list(hex_to)
        # print(f"{needed_adds} - {needed_removes} = {needed_change} -> {needed_moves}")

        for i, from_digit in reversed(list(enumerate(hex_to))):
            if hex_from[i] == from_digit:
                continue

            initial_adds = change_hex[hex_from[i]][from_digit][Hex.add]
            initial_removes = change_hex[hex_from[i]][from_digit][Hex.remove]
            initial_moves = max(initial_adds, initial_removes)

            # die adds und removes die in dieser iteration nicht geaendert werden koennen
            # also von inf change ohne diese iteration
            bare_adds = needed_adds - initial_adds
            bare_removes = needed_removes - initial_removes

            # print(f"{i} {from_digit} move {needed_moves} change {needed_change} specific a{adds} r{re
            new_digit = -1
            for to_digit in reversed(range(16)):
                iter_adds = bare_adds + change_hex[hex_from[i]][to_digit][Hex.add]
                iter_removes = bare_removes + change_hex[hex_from[i]][to_digit][Hex.remove]

                iter_change = iter_adds - iter_removes
                iter_moves = max(iter_adds, iter_removes)

                # print(f"iter {i} {to_digit} add {iter_adds} remove {iter_removes} move {iter_moves} c

                if needed_change > 0 and iter_change > needed_change:   continue
                if needed_change < 0 and iter_change < needed_change:   continue

                # print("heh", iter_adds, iter_removes)
                if iter_moves < needed_moves:
                    # print("??????????????")
                    new_digit = to_digit
                    break

            if new_digit != -1:
                break


        if new_digit == -1:
            pass
            # print_hex_list(hex_to)
            # logging.warning('No digit found to revert illegal moves (if it occurs only once, it is no
        else:   hex_to[i] = new_digit


    return hex_to


def fill_up_moves(hex_from: list, hex_to_: list, max_moves: int) -> list:
    if len(hex_from) <= 0 or len(hex_to_) <= 0:
        raise Exception('hex_from␣or␣hex_to␣is␣empty.␣Thus␣the␣problem␣is␣not␣in␣fill␣up␣moves')
    hex_to = hex_to_.copy()


    for i, digit1 in enumerate(hex_to):
        if digit1 == 15:
            continue
        hex_change = {}

        initial_change = change_hex[hex_from[i]][digit1][Hex.add] - change_hex[hex_from[i]][digit1][Hex
        for possibility in reversed(range(digit1 + 1,16)):
            possible_change = change_hex[hex_from[i]][possibility][Hex.add] - change_hex[hex_from[i]][p

            total_change = initial_change - possible_change
            if total_change not in hex_change:
                hex_change[total_change] = []
            hex_change[total_change].append(possibility)
        found_anything = False
```

```
215         for j, digit2 in reversed(list(enumerate(hex_to))[i+1:]):
                initial_change = change_hex[hex_from[i]][digit2][Hex.add] - change_hex[hex_from[i]][digit2]
217             for possibility in reversed(range(16)):
                    possible_change = change_hex[hex_from[i]][possibility][Hex.add] - change_hex[hex_from[i
219                     Hex.remove]

221                 total_change = initial_change - possible_change

223                 if -total_change in hex_change:

225                     prev_i = hex_to[i]
                        prev_j = hex_to[j]
227
                        hex_to[i] = max(hex_change[-total_change])
229                     hex_to[j] = possibility

231                     if needed_moves_between_hex(hex_from, hex_to) > max_moves:
                            hex_to[i] = prev_i
233                         hex_to[j] = prev_j

235                     else:
                            found_anything = True
237                         break

239             if found_anything:
                    break
241         if needed_moves_between_hex(hex_from, hex_to) >= max_moves:
                break
243




247
    return hex_to
249
def execute_file(file_number: int = 0) -> list:
251     print("\n\n#########################################################################################
        # read example file
253     with open(f'examples/hexmax{file_number}.txt') as f:
            hex_str, moves = f.read().splitlines()
255         moves = int(moves)
            hex_number = [int(digit, 16) for digit in hex_str]
257         print(f'Moves: {moves}')
            print(f'Hex: {hex_str}')
259
        hex_length = len(hex_str)
261     stick_count = get_sticks(hex_str)

263     print(f'hex length: {hex_length}')
        print(f'Sticks: {stick_count}')
265
        with_infinite_moves = get_biggest_hex_infinite(stick_count, hex_length)
267     print_hex_list(with_infinite_moves)

269     # get needed moves between hex numbers
        needed_moves = needed_moves_between_hex(hex_number, with_infinite_moves)
271     print(f'Needed moves: {needed_moves}')

273     # revert the moves that arent possible
        if needed_moves <= moves:
275         return with_infinite_moves

277     temp_solution = revert_illegal_moves(hex_number, with_infinite_moves, moves)
        print_hex_list(temp_solution)
279     solution = fill_up_moves(hex_number, temp_solution, moves)
        print_hex_list(solution)
281     # print(needed_value_between_hex(hex_number, solution))
        print(needed_value_between_hex(hex_number, solution))
283     return hex_number, moves,solution, needed_value_between_hex(hex_number, solution)

285

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):
    with open(f'solutions/hexmax{example}.txt', "w") as f:
        pass

    length = 20
    gap = 10
    thickness = 1
    digits = len(hex_number)
    # create pillow image
    img = Image.new('RGB', ((length + gap) * digits + gap, (length + length + gap) * (moves + 1) + gap)

    start_hex = [hex_matrix[i].copy() for i in hex_number]
    end_hex = [hex_matrix[i].copy() for i in sollution]

    save_in_text(start_hex, example)

    img = draw_digits(img, start_hex, 0, moves + 1, length=length, gap=gap, thickness=thickness)
    for i in range(moves):
        add = False
        remove = False
        to_break = False
        for j, (digit1, digit2) in enumerate(zip(start_hex, end_hex)):
            for k, (bit1, bit2) in enumerate(zip(digit1, digit2)):
                if bit1 and (not bit2) and not add:
                    add = True
                    start_hex[j][k] = bit2
                    if remove:
                        to_break = True
                        break
                elif (not bit1) and bit2 and not remove:
                    remove = True
                    start_hex[j][k] = bit2
                    if add:
                        to_break = True
                        break

            if to_break:
                break

        save_in_text(start_hex, example)
        img = draw_digits(img, start_hex, i + 1, moves + 1, length=length, gap=gap, thickness=thickness

    img.save(f'solutions/hexmax{example}.png')
```