Aufgabe 3: HexMax

Teilnahme-ID: 01048

Bearbeiter/-in dieser Aufgabe:
Lorian Linnertz Arend

21. April 2022

Inhaltsverzeichnis

Lösungsidee

GRUNDSÄTZE:
1) Die Summe aller Kontrolsummen muss am Ende 0 sein
2) Die Summe aller Umlegungen muss am Ende unter dem Maximum sein
3) Je weiter links eine Zahl ist, umso wichtiger ist sie. So macht zum Beispiel bei der Zahl 100 eine Veränderung der 0.ten Position auf 2 (also 200)einen größeren Unterschied als die restlichen auf 9 (also 199). Daraus lässt sich Schlussfolgern, dass eine Erhöhung auf 200 besser ist als die Erhöhung auf 199, egal wie viel Umlegungen man dafür braucht

Der Weg, mit Hilfe dieses Wissen auf die Größtmögliche Zahl zu kommen, ist in dem man alle Zahlen von links nach rechts überprüft ob es eine Möglichkeit gibt sie zu erhöhen

Das Hauptproblem ist dabei sicher zustellen, dass die Zahl ausgeglichen ist, also die Summe aller Kontrolsummen 0 ist. Eine Möglichkeit das sicher zustellen ist, indem man immer Paare ausgeglichener Summen sucht. Also man nimmt die Kontrollsumme, und sucht welche Kombination diese ausgleichen würde

Kombinationen finden

Um dies zu erreichen muss man in einem ersten Schritt alle einzigartigen Kombinationen suchen. Eine einzigartige Kombination ist eine Kombination welche sich nicht kürzen lässt zB. 3+3-1=5 Beispiel für eine nicht einzigartige Kombination wäre 3+2+1-1 = 5, denn hier könnte man 1-1 herauskürzen.
Vereinfach wird das ganze durch die Feststellung, dass es nur 10 Mögliche Zahlen gibt, denn die größte Kontrollsumme ist Vektor(8) und die kleinste ist Vektor(1). Wenn man jetzt

Summe(Vektor(1)-Vektor(8)) ergibt -5 welches die untere Grenze darstellt und Summe(Vektor(8)-Vektor(1)) ergibt 5 was die größte Zahl darstellt.

Kombinationen matchen

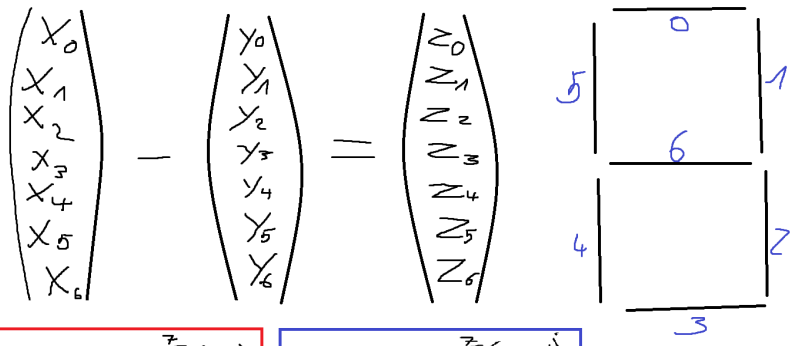
Um die Kombinationen zu matchen, muss man nun in den Verfügbaren Umwandlungen jeder Zahl nach den Kombinationen suchen, welche auf unser Bedüfnisse passen. Das Ziel ist es die Kombination zu finden, welche am wenigsten Umwandlungen benötigt und unser Kontrollsumme ausgleicht.

Zahlen anpassen

Um die Zahlen anzupassen, nehmen wir die Zahlen der Kombination, und ersetzen sie uns in unserer Liste. Entscheiden hierbei ist, dass wir die Zahl ganz links, welche wir gerade optimiert habe aus der liste löschen und von vorne Anfangen, also die jetzt linkeste Zahl, welche vorher zweit linkeste. Das machen wir so lange, bis wir alle Zahlen durch haben

(1)Position = man fängt von links an die Zahlen zu zählen. So hätten bei der Zahl 315, die 3 die Position 0, die 1 die Position 1 und die 5 hätte die Position 2

Umsetzung

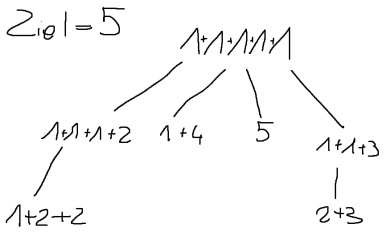


Wir erstellen uns als erstes eine Liste in der wir alle HexZahlen in Vektor Form hinschreiben. (1 = Stäbchen, 0 = kein Stäbchen)

Als nächstes nehmen wir unsere gegegene Startzahl und errechnen für jede Position, jede mögliche Umwandlung und fügen diese in eine Liste von Listen. Dabei ist der Index der jeweiligen Listen ebenfalls der Name der dazu gehörenden Position.

Nun iterieren wir durch alle möglichen Umwandlungen der Zahl ganz links.

Kombinationen finden



Von der gegebenen Kontrollsumme errechnet man nun die Kombinationen die diese Zahl ausgleichen. Dazu benutzen wir ein Baum Model. Wir fangen mit der leichtesten Kombination an was n mal die 1 wäre. Dies sind nun alle einfachen Möglichkeiten.

Um die Komplexen Kombinationen zu berechnen muss man folgende Kriterien beachten damit sie gültig sind:

-Zum einen dürfen keine zwei gleichen Zahlen mit unterschiedlichem Vorzeichen vorkommen

- Darf die Summe der entgegen gesetzten Zahlen nicht so groß sein wie irgend eine Normale Zahl

-Die Summe an normalen 1sen darf keiner entgegen gesetzten Zahl entsprechen(2)

Kombinationen matchen

Um die Kombinationen zu matchen, erstellt man für jedes in der Kombiation enthalte Zahl eine Liste. In dieser Liste stehen alle möglichen Umwandlungen sortiert nach benötigten Umwandlungen(klein->groß). Man fängt mit der kürzesten Liste an und wenn man aus dieser Liste eine Zahl entnommen worden ist, löscht man aus allen anderen Listen den mit der Position der Zahl assozierten Position.

Zahlen anpassen

Bei allen Positionen, von denen eine Zahl entnommen wurde, subtrahiert man den entnommenen Tupel aus (sum,changes) mit allen anderen enthalten Tupeln. Bei genauerer Betrachtung bemerkt man dass, das das selbe ist, als wenn man alle Zahlen wieder in Vektoren um rechnet und dass die Arbeit vor nimmt. Nach dem die Zahl ganz links fertig optimiert wurde, wird sie aus der Liste Gelöscht und in die Lösungs liste eingefügt. Nun beginnt das ganze von vorne mit einer neuen Zahl

(2)Entgegen gesetzte Zahlen sind Zahlen mit einem anderen Vorzeichen

Beispiele

--- HEXMAX0.TXT ---

Die maximale Zahl, welche man mit 3 Umlegungen und der Ausgangszahl "D24" erreichen kann ist:
EE4

0. Umlegung: Das Stäbchen rechts oben von der 0.Zahl wird zur 0.Zahl, oben umgelegt
1. Umlegung: Das Stäbchen rechts unten von der 0.Zahl wird zur 0.Zahl, links oben umgelegt
2. Umlegung: Das Stäbchen rechts oben von der 1.Zahl wird zur 1.Zahl, links oben umgelegt

--- HEXMAX1.TXT ---

Die maximale Zahl, welche man mit 8 Umlegungen und der Ausgangszahl "509C431B55" erreichen kann ist:
FFFF4D1B55

0. Umlegung: Das Stäbchen rechts unten von der 0.Zahl wird zur 0.Zahl, links unten umgelegt
1. Umlegung: Das Stäbchen unten von der 0.Zahl wird zur 1.Zahl, in der Mitte umgelegt
2. Umlegung: Das Stäbchen rechts oben von der 1.Zahl wird zur 2.Zahl, links unten umgelegt
3. Umlegung: Das Stäbchen rechts unten von der 1.Zahl wird zur 3.Zahl, in der Mitte umgelegt
4. Umlegung: Das Stäbchen unten von der 1.Zahl wird zur 5.Zahl, links unten umgelegt

--- HEXMAX2.TXT ---

Die maximale Zahl, welche man mit 37 Umlegungen und der Ausgangszahl "632B29B38F11849015A3BCAEE2CDA0BD496919F8" erreichen kann ist:
FFFFFFFFFFFFFFFFFFFDECAEE2CDA0BD49B919F8

0. Umlegung: Das Stäbchen rechts unten von der 0.Zahl wird zur 0.Zahl, oben umgelegt
1. Umlegung: Das Stäbchen unten von der 0.Zahl wird zur 1.Zahl, links unten umgelegt
2. Umlegung: Das Stäbchen rechts oben von der 1.Zahl wird zur 1.Zahl, links oben umgelegt
3. Umlegung: Das Stäbchen rechts unten von der 1.Zahl wird zur 2.Zahl, links oben umgelegt
4. Umlegung: Das Stäbchen unten von der 1.Zahl wird zur 3.Zahl, oben umgelegt
5. Umlegung: Das Stäbchen rechts oben von der 2.Zahl wird zur 4.Zahl, links oben umgelegt
6. Umlegung: Das Stäbchen unten von der 2.Zahl wird zur 5.Zahl, links unten umgelegt

7. Umlegung: Das Stäbchen rechts unten von der 3.Zahl wird zur 6.Zahl, oben umgelegt
8. Umlegung: Das Stäbchen unten von der 3.Zahl wird zur 7.Zahl, links unten umgelegt
9. Umlegung: Das Stäbchen rechts oben von der 4.Zahl wird zur 7.Zahl, links oben umgelegt
10. Umlegung: Das Stäbchen unten von der 4.Zahl wird zur 10.Zahl, oben umgelegt
11. Umlegung: Das Stäbchen rechts oben von der 5.Zahl wird zur 10.Zahl, links unten umgelegt
12. Umlegung: Das Stäbchen rechts unten von der 5.Zahl wird zur 10.Zahl, links oben umgelegt
13. Umlegung: Das Stäbchen unten von der 5.Zahl wird zur 10.Zahl, in der Mitte umgelegt
14. Umlegung: Das Stäbchen rechts unten von der 6.Zahl wird zur 11.Zahl, oben umgelegt
15. Umlegung: Das Stäbchen unten von der 6.Zahl wird zur 11.Zahl, links unten umgelegt
16. Umlegung: Das Stäbchen rechts oben von der 7.Zahl wird zur 11.Zahl, links oben umgelegt
17. Umlegung: Das Stäbchen rechts unten von der 7.Zahl wird zur 11.Zahl, in der Mitte umgelegt
18. Umlegung: Das Stäbchen unten von der 7.Zahl wird zur 13.Zahl, oben umgelegt
19. Umlegung: Das Stäbchen rechts oben von der 8.Zahl wird zur 13.Zahl, links unten umgelegt
20. Umlegung: Das Stäbchen rechts unten von der 8.Zahl wird zur 14.Zahl, links unten umgelegt
21. Umlegung: Das Stäbchen unten von der 8.Zahl wird zur 15.Zahl, in der Mitte umgelegt
22. Umlegung: Das Stäbchen rechts oben von der 10.Zahl wird zur 16.Zahl, oben umgelegt
23. Umlegung: Das Stäbchen rechts unten von der 10.Zahl wird zur 16.Zahl, links unten umgelegt
24. Umlegung: Das Stäbchen rechts oben von der 11.Zahl wird zur 16.Zahl, links oben umgelegt
25. Umlegung: Das Stäbchen rechts unten von der 11.Zahl wird zur 16.Zahl, in der Mitte umgelegt
26. Umlegung: Das Stäbchen rechts oben von der 12.Zahl wird zur 17.Zahl, links unten umgelegt
27. Umlegung: Das Stäbchen rechts unten von der 12.Zahl wird zur 19.Zahl, links unten umgelegt
28. Umlegung: Das Stäbchen unten von der 12.Zahl wird zur 20.Zahl, oben umgelegt

--- HEXMAX3.TXT ---

Die maximale Zahl, welche man mit 121 Umlegungen und der Ausgangszahl
"0E9F1DB46B1E2C081B059EAF198FD491F477CE1CD37EBFB65F8D765055757C6F4796BB8B3DF7FCAC606DD0627D6B48C17C09"
erreichen kann ist:
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDF7FFACB0BDD0B27DBB48C17C09

--- HEXMAX4.TXT ---

Die maximale Zahl, welche man mit 87 Umlegungen und der Ausgangszahl
"1A02B6B50D7489D7708A678593036FA265F2925B21C28B4724DD822038E3B4804192322F230AB7AF7BDA0A61BA7D4AD8F888" erreichen kann ist:
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF4DD8F2038E3B4804192322F230AB7AF7BDA0AB1BA7D4AD8F888

--- HEXMAX5.TXT ---

Die maximale Zahl, welche man mit 1369 Umlegungen und der Ausgangszahl
"EF50AA77ECAD25F5E11A307B713EAAEC55215E7E640FD263FA529BBB48DC8FAFE14D5B02EBF792B5CCBBE9FA1330B867E330A6412
870DD2BA6ED0DBCAE553115C9A31FF350C5DF993824886DB5111A83E773F23AD7FA81A845C11E22C4C45005D192ADE68AA9AA57406
EB0E7C9CA13AD03888F6ABEDF1475FE9832C66BFDC28964B7022BDD969E5533EA4F2E4EABA75B5DC11972824896786BD1E4A7A7748
FDF1452A5079E0F9E6005F040594185EA03B5A869B109A283797AB31394941BFE4D38392AD12186FF6D233585D8C820F197FBA9F6F063A
0877A912CCBDCB14BEECBAEC0ED061CFF60BD517B6879B72B9EFE977A9D3259632C718FBF45156A16576AA7F9A4FAD40AD8BC87EC
569F9C1364A63B1623A5AD559AAF6252052782BF9A46104E443A3932D25AAE8F8C59F10875FAD3CBD885CE68665F2C826B1E1735EE2F
DF0A1965149DF353EE0BE81F3EC133922EF43EBC09EF755FBD740C8E4D024B033F0E8F3449C94102902E143433262CDA1925A2B7FD01B
EF26CD51A1FC22EDD49623EE9DEB14C138A7A6C47B677F033BDEB849738C3AE5935A2F54B99237912F2958FDFB82217C175448AA8230
FDCB3B3869824A826635B538D47D847D8479A88F350E24B31787DFD60DE5E260B265829E036BE340FFC0D8C05555E75092226E7D54DEB
42E1BB2CA9661A882FB718E7AA53F1E606" erreichen kann ist:
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEAE217C175448AA8230FDCB3B
38B9824A82BB35B538D47D847D8479A88F350E24B31787DFDB0DE5E2B0B2B5829E03BBE340FFC0D8C05555E7509222BE7D54DEB42E1
BB2CA9BB1A882FB718E7AA53F1EB0B

Quellcode

```
hexList = [[np.matrix("1 1 1 1 1 1 0"),"0"],[np.matrix("0 1 1 0 0 0 0"),"1"], [np.matrix("1 1 0 1 1 0 1"),"2"], [np.matrix("1 1 1 1 0 0 1"),"3"],
[np.matrix("0 1 1 0 0 1 1"),"4"],
[np.matrix("1 0 1 1 0 1 1"),"5"], [np.matrix("0 0 1 1 1 1 1"),"6"], [np.matrix("1 1 1 0 0 0 0"),"7"], [np.matrix("1 1 1 1 1 1 1"),"8"], [np.matrix("1 1 1 1 0 1 1"),"9"],
[np.matrix("1 1 1 0 1 1 1"),"A"], [np.matrix("0 0 1 1 1 1 1"),"B"], [np.matrix("1 0 0 1 1 1 0"),"C"], [np.matrix("0 1 1 1 1 0 1"),"D"], [np.matrix("1 0 0 1 1 1 1"),"E"], [np.matrix("1 0 0 0 1 1 1"),"F"]
]
```

```
available_nums = [[-1,-2,-3,-4,-5],[1,2,3,4,5]]

def get_compensations(balance_num,available_nums = available_nums):
    #Berechne die einfachen Kombinationen
    balance_num = balance_num * (-1) #Wandelt aus einer positiven Zahl eine neagtiv und umgedreht Grundlage=> balance_num +
get_compensations(available_nums,balance_num) = 0 <=> get_compensations(available_nums,balance_num) = - balance_num
    combination_list = []
    combination_list.extend(get_combinations(available_nums[1],balance_num))
    #Berechne max_value
    max_opposite = abs(balance_num) -2
    if max_opposite < 0:
        max_opposite = 0
    #Berechne die komplexen Kombinationen
    for i in range(1,max_opposite+1):
        opposite_combinations= get_combinations(available_nums[1],i*(abs(balance_num)/balance_num)*(-1))
        regular_combinations= get_combinations(available_nums[1],balance_num+i*(abs(balance_num)/balance_num))
#abs(balance_num)/balance_num -> falls balance_num negaitv ist so kommt -1 raus sonst 1
        for combi in opposite_combinations:
            x = combination_filter(copy.deepcopy(combi),copy.deepcopy(regular_combinations))
            for item in x:
                item.extend(combi)
                combination_list.append(item)
    return combination_list
def combination_filter(reference_combi,combinations):
    sum = 0 #setzt den Zähler auf 0
    for i in reference_combi: #iteriert durch jede Zahl von reference_combi
        i = i * (-1) #Da reference_combi das entgegen gesetzt Vorzeichen zu den combinations hat, wir das Vorzeichen angepasst um besser vergleichen
zu könnnen
        sum = sum + i
    indexes_to_remove = []
    for i,combi in enumerate(combinations): #iteriert durch alle möglichen Kombinationen ->combi stellt eine Kobination aus der vorherigen liste
regular_combinations dar
        for element in reference_combi: #iteriert durch alle elemente der reference_combi
            element = element * (-1) #In diesem Schritt wird das Vorzeichen von Element getauscht
            if element in combi or sum in combi or abs(combi.count(abs(element)/element)) >= abs(element): #Die if-Abfrage, überprüft 3 Sachen: 1.Ist
element(Vorzeichen geändert) bereits in combi, 2. Ist die Summe(Vorzeichen angepasst) des opposite_combination in combi(-> Falls die zutrifft, so
hätte man in der Kompination ein eigenständiges paar, was verhindert werden soll), 3. ist die summe der 1sen in combi größer oder gleich
element(dies würde wieder zu einem eigenständigem paar führen)
                indexes_to_remove.append(i) #Falls eine der drei bedingungen Zutrifft, so wird der index dieser Combi in eine Liste eingefügt und später
entfernt

    indexes_to_remove = list(set(indexes_to_remove)) #Falls mehrere male der gleiche Index vorhanden ist, so werden alle außer einer gelöscht
    indexes_to_remove.sort(reverse=True) #sortiert die Indexe von groß nach klein, so dass sich die Indexe beim löschen nicht verschieben
    for i in indexes_to_remove:
        combinations.pop(i)
    return combinations
def get_combinations(list_num,goal): #soll alle kombinationsmöglichkeiten zurück geben, welche alle die gleichen
    target = abs(goal)
    combinations = []
    origin_combination = [] #Auf dieser Kombination baut die ganze Funktion auf
    for _ in range(int(abs(target))): #Definiert die Ausgangskombination,welche ebenfalls eine gültige kombination ist
        origin_combination.append(1)
    combinations.append(origin_combination)
    memory = [origin_combination] #In dieser Variable, werden die neuen Kombinationen zwischen gespeichert um überprüft zu werden. Die die
bereits überprüft worden sind werden wieder entfernt
    while len(memory)>0:#Die while-Schleife soll solange laufen, bis der "Tree" fertig ausgebaut wurde und in memory keine noch zu überprüfenden
Wege mehr da sind
        for i in list_num[1:]: #iteriert durch alle zur Verfügung stehenden Zahlen außer 1
            if memory[0] == []:#Falls memory leer sein sollt,so wird diese schleife abgebrochen
                break
            if memory[0].count(1) >= i: #Falls die Anzahl an 1sen ausreicht um i zuerstellen,dann
                new_combination = memory[0] #speichere in new_combination den Inhalt von memory[0]
                new_combination = new_combination[:(i*(-1))] #Entferne i 1sen
                new_combination.append(i) #ersetze sie durch 1
                new_combination.sort(reverse=True) #Nun wird noch mal alles von Groß nach klein sortiert
                if new_combination not in combinations: #Falls es die Kombination nicht bereits geben sollte, dann
                    memory.append(new_combination) #füge sie zu memory hinzu um sie überprüfen zu lassen
                    combinations.append(new_combination) #füge die Kombination hinzu
        memory.pop(0) #Anschließend wir die gerade überprüfte Kombination wieder gelöscht
    for i,list in enumerate(combinations): #Sortiert wieder alles von klein nach groß
        list.sort()
        combinations[i] = list
    if float(abs(goal)/goal) == 1: #überprüft op goal postiv ist Ansatz: abs(goal)/goal = 1 -> bedeutet postiv;   abs(goal)/goal = -1 -> bedeutet negativ
```

```
        return combinations
    else: #Falls die Zahl negativ sein sollte, so wird jedes element mit -1 multipliziert
        for i,combi in enumerate(combinations):
            for j,item in enumerate(combi):
                combinations[i][j] = item * (-1)
        return combinations
def Find_Max(list):#input: [[index,value]]
    maxIndex=0              #Definiert die Variable x als int oder float
    maxNum = list[maxIndex][1]    #Die Variable maxNum,bekommt als Startwert die erste Zahl aus einer gegebenen Liste, diese wird dann immer
durch die nächst grössere Zahl ersetzt, solange bis keine grössere Zahl mehr in der Liste ist.
    for i,b in enumerate(list): #iteriert durch alle Elemente der gegebenen Liste durch
        if maxNum is None:#Falls die aktull größte Zahl null sein sollte, so wird sie ersetzt
            maxNum = b[1]    #Falls eine grössere Zahl als maxNum gefunden wird, so wird maxNum durch diese Zahl ersetzt
            maxIndex = i       #ebenfalls wird der Index dieser neuen grössten Zahl aktualisiert
            continue
        elif b[1] is None: #Falls b[1] None sein sollte so wird sie übersprungen
            continue
        elif (b[1] > maxNum):   #Vergleicht ob das Element grösser ist als die bislang grösste gefundene Zahl
            maxNum = b[1]    #Falls eine grössere Zahl als maxNum gefunden wird, so wird maxNum durch diese Zahl ersetzt
            maxIndex = i       #ebenfalls wird der Index dieser neuen grössten Zahl aktualisiert
    return maxIndex   #gibt maxIndex zurück
def SortIndexes_MinToMax(list): #input: [[index,value][...]]#Erstellt eine neue Liste mit den Indexen der Ausgangsliste sortiert von dem kleinsten
Wert bei [0] und dem größtem am Ende
    index = Find_Max(list)
    maxValue = list[index][1]
    sortedIndexs=[list[index][0]] #setzt einen ersten ReferenzWert ein um fest
    sortedValues = [list[index][1]]
    sorted_list = [list[index]]
    list.pop(index)
    none_indexes = []
    none_values = []
    none_list = []
    for item in list: #Da der Refernz Wert den Index 0 hatte so wird dieser hier übersprungen
        if item[1] is None:
            none_indexes.append(item[0])
            none_values.append(item[1])
            none_list.append(item)
            continue
        elif item[1] == maxValue:
            sortedIndexs.append(item[0])
            sortedValues.append(item[1])
            sorted_list.append(item)
        for i,value in enumerate(sortedValues):
            if item[1] < value: #Falls der Wert kleiner ist alls value, so wird die Zahl vor Value eingefügt
                sortedIndexs.insert(i,item[0])
                sortedValues.insert(i,item[1])
                sorted_list.insert(i,item)
                break
    sortedIndexs.extend(none_indexes)
    sortedValues.extend(none_values)
    sorted_list.extend(none_list)
    return sorted_list



#------------------------------------------------------------------------------

class Num_Max():
    def __init__(self,num,changes,meta_list):
        self.num = num
        self.changes = changes
        self.meta_list = meta_list
        self.adjusted_meta_list = copy.deepcopy(self.meta_list)
        self.maximum_num = Num_Max.main(self)

    def update_meta_data(self,set_with_position):#Updatet die Werte mit dem neuen Set, ein set_with_position ist eine Liste mit der Form
[sum,changes,Position]
        position = set_with_position[2]
        meta_data = self.adjusted_meta_list[position]
        for i,meta_num in enumerate(meta_data):
            self.adjusted_meta_list[position][i][2][0] = meta_num[2][0] - set_with_position[0]
            self.adjusted_meta_list[position][i][2][1] = meta_num[2][1] - set_with_position[1]
```

```
        for pos in different_nums_positions: #iteriert durch die bereits fertige gestellt Liste different_nums_positions -> In dieser Liste wird für jede position die jeweiligen Zahlen gespeichert
            for i,num in enumerate(different_nums): #iteriert durch different_nums PS: Da beide Listen gleich sortiert sind, sind die Indexe von amount_different_nums und different_nums gleich
                if num in pos: #Falls die Zahl in der bestimmten position vorhanden ist, so erhöhe um 1
                    amount_different_nums[i][1] = amount_different_nums[i][1] +1


        return different_nums, different_nums_positions,amount_different_nums
        #for pos in different_nums_positions:

    def main(self):
        max_num = [] #In diese Variable sollen die neuen Maximalen Zahlen gespechert werden
        for i in range(len(meta_list)): #ruft den Code so oft auf, wie es Positionen gibt. Da die Funktion Num_Max.transform(self) die Länge von der adjusted_meta_list immer um 1 verringert, wird diese am Ende eine leere Liste sein
            print(f"{round((i+1)/len(meta_list)*100)}%\t {i+1}/{len(meta_list)}")
            max_num.append(Num_Max.transform(self)[0])
        return max_num

    def transform(self): #In diese Funktion kommt das Auswahlverfahren, für adjusted_meta_list[0]
        for num in list(reversed(self.adjusted_meta_list[0])):
            if num[2][1] == 0: #Diese IF-Abfrage,soll verhindern, dass die Zahl kleiner wird
                self.adjusted_meta_list.pop(0)
                return num
            elif self.changes < num[2][1]: #überprüft ob man mehr Umlegungen benötigt als zur Verfügung stehen um die Zahl zu transformieren
                continue
            self.changes = self.changes -num[2][1]
            is_possible,valid_combination = Num_Max.combination_filter(self,num[2][0])

            if is_possible and valid_combination == None:#überprüft, ob der Fall zutriff, in welchem man keine anderen Wert updaten muss, Grund: wenn num[2][0] == 0 ist, so ist diese Zahl bereits ausgeglichen
                self.adjusted_meta_list.pop(0)
                return num
            elif is_possible:
                self.changes = self.changes - valid_combination[1]
                for set_with_position in valid_combination[0]:
                    Num_Max.update_meta_data(self,set_with_position)
                self.adjusted_meta_list.pop(0)
                return num
            else:
                self.changes = self.changes + num[2][1] #Falls die Umformung nicht möglich ist, so addiere die verbrauchten Umlegungen(welche wir vorher abgezogen hatte) wieder drauf, da wir sie nicht benötigt haben

        return True #Falls, ein passendes Set gefunden wurde, dann wird True zurückgegebene


    def get_best_positions(self,values,combination): #values is a list
        values = copy.deepcopy(values)
        combination = copy.deepcopy(combination)
        delta_meta_list = []
        for pos in copy.deepcopy(self.adjusted_meta_list[1:]): #Die Code erstellt eine Kopie von adjusted_meta_list ab dem index[1], da index[0] die Position ist die wir gerade versuchen zu optimieren. Es werden zum einen nur die benötigten werte kompiert und zum anderen wird aus der for [hex_Num,dez_Num,[sum,changes]] -> [sum,changes]
            pos_list = []
            for item in pos:
                item = item[2]
                for val in values:
                    if val == item[0]:
                        pos_list.append(item)
                        break
            if pos_list != []:
                delta_meta_list.append(pos_list)


        for i,pos in enumerate(delta_meta_list): #sortiert die Liste so, dass für die jeweilige position immer bei pos[0] das Element mit dem niedrigsten change steht, eben falls, falls ein Value mehrmals vorhanden sein sollte, so bleibt nur der mit dem niedrigsten changes erhalten
            vals = copy.deepcopy(values) #erstelle eine Kopie der Values
            pos = SortIndexes_MinToMax(pos)
            for j,item in enumerate(pos): #iteriert durch die neue sortierte Liste von vorne nach hinten
                if item[0] in vals: #überprüft ob es den Value bereits vorher und damit mit niedrigerem Change gibt,falls dies zutriff,so wird das item entfernt
                    vals.remove(item[0])
                else:
```

```
                pos[j] = []
            while [] in pos:
                pos.remove([])
            delta_meta_list[i] = pos

        list_possible_values = []
        for v in values:#erstellt die Strucktur für die Liste list_possible_values
            list_possible_values.append([])

        for i,pos in enumerate(delta_meta_list): #Verwandelt die Kopie von adjusted_meta_list welche nach den Positionen strukturier ist, in eine neue liste,welche nach den verschiedenen Values struktueriert ist
            for item in pos:
                item.append(i)
                for j,v in enumerate(values):
                    if item[0] == v:
                        list_possible_values[j].append(item)
        for i,list in enumerate(list_possible_values):# sortiert die Liste so, dass für list_possible_values[i][0] gilt, dass es der values[i] mit dem insgesamt niedrigstem chang ist
            if list == []: #überprüft, ob die Liste nicht leer ist. Falls sie leer sein sollte, dann bedeutet das, dass der Benötigte wert in keiner der Postitionen zu finden ist
                return [None,None] #gibt eine Liste mit [None,None] zurück.
            list_possible_values[i] = SortIndexes_MinToMax(list)
        new_list = []
        for i,item in enumerate(list_possible_values): #Erstellt eine neue Liste von Listen mit der Form [values[i],len(list[i]),list[i]] -> Diese neue Liste wird erstellt, damit man die Funktion SortIndexes_MinToMax() benutzen kann
            new_list.append([values[i],len(item),item])
        if new_list == []:
            return [None,None] #gibt eine Liste mit [None,None] zurück.
        list_possible_values = SortIndexes_MinToMax(new_list) #Die neu optimierte Liste sieht wie folgt aus, [[value,lenght,list],...] -> sortiert nach der länge, damit wir autmatisch mit dem Value anfangen, welcher in den wenigsten Positionen vorkommt

        used_positions = [] #Hier sollen, die namen der bereits benutzten Position stehen, damit eine Position nicht mehrmals verwendet wird
        result = []
        sum_changes = 0

        for item in list_possible_values: #iteriert durch die neu struktuerierte
            for _ in range(combination.count(item[0])): #Zählt wie oft der ausgewählt wert in der Variable combinations steht und lässt den folgenden Code so oft ausführen
                control_bool = False #Diese Variable soll später helfen zu überprüfen ob eine lösung gefunden wurde
                for data in item[2]:
                    if data[2] not in used_positions: #Kontrolliert zum einen ob die Position von data nicht bereits verwendet wurde
                        used_positions.append(data[2]) #Füge die gerade benutzte Position zu der Variable used_positions
                        result.append(data)
                        sum_changes += data[1] #summiert die anzahl der Changes auf
                        control_bool = True
                        break
                if control_bool == False: #Falls kein passender data_set in item gefunden wurde, so gibt None zurück, was bedeutet, dass es keine Lösung gibt
                    return [None,None] #gibt eine Liste mit [None,None] zurück. Grund: Da es weder ein Valides Result noch eine Summe von Changes
            return [result,sum_changes] #gibt das Resultat sowie die Summe der benötigten änderungen


        return values, list_possible_values

    def combination_filter(self,balance_num): #Diese Methode soll die unnötigen Kombinationen raus werfen und die Kombinationen der länge nach sortieren
        if balance_num == 0:
            return True,None
        combination_list = get_compensations(balance_num)
        valid_combinations = []
        for combi in combination_list:
            combination = Num_Max.get_best_positions(self,list(set(combi)),combi)
            valid_combinations.append(combination)
        valid_combinations = SortIndexes_MinToMax(valid_combinations) #sortiert alle möglichkeiten,so dass die mit den insgesamt wenigsten Changes beim Index 0 steht
        if valid_combinations[0] == None or valid_combinations[0][1] == None:
            return False,None
        elif valid_combinations[0][1] <= self.changes:
            valid_combinations[0][0][0][2] += 1 #Der Name(Index der Position), wird um 1 erhöht, da wir vorher 0 übersprung hatten(Wir hatten 0 übersprungen, da es adjusted_meta_list[0] und wir diese Optimieren wollen)
            return True,valid_combinations[0]
        else:
            return False,None
```

```
def get_meta_nums(list,changes): #Diese Funktion soll eine Liste zurückgeben, in welcher alle Metadaten, für eine Mögliche umwandlung enthalten sind
    meta_list = [] #In diese Liste sollen später alle in jeweils eigenen Listen die Metadaten für die einzelnen Positionen stehen, die Position entspricht dem Index
    for num_vec in list: #Iteriert durch die Vektoren der gegebenen Zahlen
        meta_data = [] #Diese Variable dient als zwischen speicher
        for i,hex_vec in enumerate(hexList): #Iteriert durch alle möglichkeiten zum umwandeln
            dif_vec = np.subtract(hex_vec[0],num_vec) #subtrahiert beide Matrizen, das Resultat ist eine Matrix, in welcher eine 1 bedeutet, dass ein Stäbchen entfernt werden muss und ein -1 bedeutet, dass dort eins fehlt
            a = analyze_vector(dif_vec)
            meta_data.append([hex_vec[1],i,a]) #Die Liste hexVec wird erweitert und hat nun die Form [Hex Verion der Zahl, Dezimal Version, [Summe,Benötigte Umlegungen]]
        meta_list.append(meta_data)
    return meta_list

def analyze_vector(vec):
    abs_sum = 0#Berechnet die Summe aller Qudrate der Elemente im gegebenen Vektor, Ziel: die Berechnung aller Verschiebungen
    for i in range(0,7):
        abs_sum += abs(vec.item(i))
    abs_sum = abs_sum / 2 #Teil das Resultat durch zwei, weil sqrsum jede einzelne Aktion angibt, als aufheben und ablegen sind jeweils einzelne Aktionen, deswegen wird das Resultat durch zwei geteilt
    sum = vec.sum()
    return [sum,abs_sum] #Diese Funktion analysiert den Differenzen Vektor und gibt die bnötigten Metadaten zurück
```