Aufgabe 5

Lösungsidee

Hier hatte ich die Idee, ähnlich wie bei der Umrechnung von Dezimal- zu Binärzahlen, rekursiv immer das größtmögliche Gewicht, das kleiner als das Zielgewicht ist, zuzufügen. Um jedoch auch noch die Vorteile der zweiseitigen Belastung der Waage zu nutzen, ist zu dem größtmöglichen Gewicht, das kleiner als das Zielgewicht ist, auch noch eine Kombination zweier Gewichte (eins links, eins rechts) hinzugekommen.

Umsetzung

Dazu habe ich eine Klasse `Scale` angelegt, die die verfügbaren Gewichte in einem `dict` speichert, mit dem jeweiligen Gewicht als Schlüssel und der Anzahl als Wert. Ein weiteres Attribut ist `possible_weights`, das zu jedem gesuchten Gewicht, also `range(10, 10010, 10)` in Python eine Liste zuordnet, die die verbleibende Differenz, die Gewichte auf der linken und die Gewichte auf der rechten Seite enthält. Dazu wird in `find_combination` zu jedem gegebenen Gewicht die Kombination ermittelt, die möglichst nah an das gewünschte Ergebnis kommt. Das wird erreicht, indem zuerst geprüft wird, ob das Gewicht durch ein einzelnes Gewichtsstück dargestellt werden kann. Wenn nicht, wird eine Kombination von zwei Gewichten aus allen verfügbaren Gewichtsstücken gesucht, die möglichst nah an das Gewicht herankommt. Dann wird das größtmögliche einzelne Gewichtsstück ermittelt, das nicht schwerer als das Zielgewicht ist. Aus den beiden wird die Version gewählt, die näher am Gleichgewicht der Waage ist. Mit dem verbleibenden Rest und dem derzeitigen Stand der Kombination und verbleibenden Gewichte ruft sich die Methode wieder selbst auf, bis eine möglichst optimale Lösung gefunden wurde. Wird ein negatives Gewicht übergeben, werden die Referenzen zur linken und rechten Sammlung von Gewichten einfach vertauscht und es wird wie gehabt verfahren.

Beispiele

Die Beispiele geben zu lange Lösungen aus, als dass diese sinnvoll dargestellt werden könnten. Ich werde einzelne, interessante Kombinationen hervorheben.

```
$ python3 aufgabe5/aufgabe5.py Beispiele/a5-
Marktwaage/beispieldaten/gewichtsstuecke0.txt
10g:
    links:[]
    rechts:[10]
20g:
    links:[]
    rechts:[10, 10]
30g:
    links:[10, 10]
```

```
    rechts:[50]
...
9940g (+10g):
    links:[]
    rechts:[5000, 1000, 1000, 1000, 500, 500, 500, 100, 100, 100, 50, 50, 10, 10,
10]
```

Bei der ersten Beispielaufgabe konnten alle Gewichte dargestellt werden, bis ihre Summe niedriger war als das gesuchte Gewicht.

```
$ python3 aufgabe5/aufgabe5.py Beispiele/a5-
Marktwaage/beispieldaten/gewichtsstuecke1.txt
10g (+10g):
    links:[]
    rechts:[]
20g (+20g):
    links:[]
    rechts:[]
30g (+30g):
    links:[]
    rechts:[]
40g (+40g):
    links:[]
    rechts:[]
50g (+8g):
    links:[]
    rechts:[42]
...
9460g:
    links:[371, 127]
    rechts:[2000, 2000, 2000, 2000, 2000, 42]
```

Bei der zweiten Aufgabe konnten die wenigsten Gewichte genau dargestellt werden, eine Ausnahme dazu bildet z.B. 9460g

```
$ python3 aufgabe5/aufgabe5.py Beispiele/a5-
Marktwaage/beispieldaten/gewichtsstuecke2.txt
10g:
    links:[]
    rechts:[10]
20g:
    links:[]
    rechts:[20]
30g:
    links:[10]
    rechts:[40]
40g:
    links:[]
    rechts:[40]
50g:
    links:[]
    rechts:[40, 10]
60g:
    links:[20]
    rechts:[80]
70g:
```

```
    links:[10]
    rechts:[80]
80g:
    links:[]
    rechts:[80]
90g:
    links:[]
    rechts:[80, 10]
100g:
    links:[]
    rechts:[80, 20]
...
9670g:
    links:[10]
    rechts:[5120, 2560, 1280, 640, 80]
9680g:
    links:[]
    rechts:[5120, 2560, 1280, 640, 80]
9690g:
    links:[]
    rechts:[5120, 2560, 1280, 640, 80, 10]
9700g:
    links:[]
    rechts:[5120, 2560, 1280, 640, 80, 20]
9710g:
    links:[40, 10]
    rechts:[5120, 2560, 1280, 640, 160]
9720g:
    links:[40]
    rechts:[5120, 2560, 1280, 640, 160]
...
9970g:
    links:[]
    rechts:[5120, 2560, 1280, 640, 320, 40, 10]
9980g:
    links:[20]
    rechts:[5120, 2560, 1280, 640, 320, 80]
9990g:
    links:[10]
    rechts:[5120, 2560, 1280, 640, 320, 80]
10000g:
    links:[]
    rechts:[5120, 2560, 1280, 640, 320, 80]
```

Bei diesem Satz von Gewichten konnten alle Gewichte exakt dargestellt werden. Das ist auch keine Überraschung, da dieser Satz von Gewichten die Binärzahlen 0000 0000 01 bis 1111 1111 11 darstellen kann, da die Gewichte die Zahlen 2^0, 2^1, 2^2, … 2^9 mit dem Faktor 10 multipliziert sind. Das ist vorteilhaft, da unsere Gewichte ebenfalls in 10er Schritten verlaufen. Da die Dezimalzahl 1.000 kleiner ist als die Binärzahl 1111 1111 11, kann sie (und jede Zahl zwischen ihr und 0) von unseren Gewichten (*10) dargestellt werden.

```
$ python3 aufgabe5/aufgabe5.py Beispiele/a5-
Marktwaage/beispieldaten/gewichtsstuecke3.txt
10g:
    links:[]
    rechts:[10]
```

```
20g:
    links:[10]
    rechts:[30]
30g:
    links:[]
    rechts:[30]
40g:
    links:[]
    rechts:[30, 10]
50g:
    links:[30, 10]
    rechts:[90]
60g:
    links:[30]
    rechts:[90]
70g (-10g):
    links:[10]
    rechts:[90]
80g:
    links:[10]
    rechts:[90]
90g:
    links:[]
    rechts:[90]
100g:
    links:[]
    rechts:[90, 10]
...
9900g:
    links:[90]
    rechts:[7290, 2430, 270]
9910g:
    links:[90]
    rechts:[7290, 2430, 270, 10]
9920g:
    links:[90, 10]
    rechts:[7290, 2430, 270, 30]
9930g (+20g):
    links:[30, 10]
    rechts:[7290, 2430, 270]
9940g (+10g):
    links:[30, 10]
    rechts:[7290, 2430, 270]
9950g:
    links:[30, 10]
    rechts:[7290, 2430, 270]
9960g:
    links:[30]
    rechts:[7290, 2430, 270]
9970g (-10g):
    links:[10]
    rechts:[7290, 2430, 270]
9980g:
    links:[10]
    rechts:[7290, 2430, 270]
9990g:
    links:[]
    rechts:[7290, 2430, 270]
```

```
10000g:
    links:[]
    rechts:[7290, 2430, 270, 10]
```

Die meisten Gewichte können mit diesem Satz annähernd beschrieben werden. Der Satz entspricht den Potenzen mit der Basis 3 mit dem Exponenten 0 bis 6, multipliziert mit 10.

```
$ python3 aufgabe5/aufgabe5.py Beispiele/a5-
Marktwaage/beispieldaten/gewichtsstuecke4.txt
10g (+2g):
    links:[21]
    rechts:[29]
20g (+4g):
    links:[5]
    rechts:[21]
30g (+1g):
    links:[]
    rechts:[29]
40g (+3g):
    links:[21]
    rechts:[29, 29]
50g:
    links:[]
    rechts:[29, 21]
60g (+2g):
    links:[]
    rechts:[29, 29]
70g (-4g):
    links:[5]
    rechts:[29, 29, 21]
80g (+3g):
    links:[287, 29]
    rechts:[399]
90g (+1g):
    links:[287, 21]
    rechts:[399]
100g (-4g):
    links:[287, 21]
    rechts:[399, 5]
...
2420g (+3g):
    links:[399, 259, 29, 5]
    rechts:[2993, 29, 29]
2430g (+3g):
    links:[399, 259, 29]
    rechts:[2993, 29, 29, 5]
2440g (+2g):
    links:[399, 259, 29, 5]
    rechts:[2993, 29, 29, 21]
2450g (+2g):
    links:[399, 259, 29]
    rechts:[2993, 29, 29, 21, 5]
2460g (+1g):
    links:[399, 259, 29, 5]
    rechts:[2993, 29, 29, 21, 21]
2470g (+4g):
    links:[399, 29, 259]
```

```
    rechts:[2993, 29, 29, 287, 5]
2480g (-1g):
    links:[399, 29, 259]
    rechts:[2993, 29, 29, 287]
2490g (+1g):
    links:[399, 29, 5]
    rechts:[2993, 29, 29, 21]
2500g (+2g):
    links:[399, 29]
    rechts:[2993, 29, 29, 5]
2510g (+2g):
    links:[399, 29, 5]
    rechts:[2993, 29, 29]
2520g:
    links:[399, 29, 5]
    rechts:[2993, 29, 21]
```

Bei diesem Satz von Gewichten können die meisten Gewichte bis auf 5 Gramm nah beschrieben werden.

```
$ python3 aufgabe5/aufgabe5.py Beispiele/a5-
Marktwaage/beispieldaten/gewichtsstuecke5.txt
10g (+10g):
    links:[]
    rechts:[]
20g (+9g):
    links:[]
    rechts:[11]
30g (-1g):
    links:[99480]
    rechts:[99511]
40g (+9g):
    links:[99480]
    rechts:[99511]
50g (+8g):
    links:[99480]
    rechts:[99511, 11]
60g (+18g):
    links:[99480]
    rechts:[99511, 11]
70g (+28g):
    links:[99480]
    rechts:[99511, 11]
80g (+38g):
    links:[99480]
    rechts:[99511, 11]
90g (+48g):
    links:[99480]
    rechts:[99511, 11]
100g (+58g):
    links:[99480]
    rechts:[99511, 11]
1
...
9900g (+9858g):
    links:[99480]
    rechts:[99511, 11]
```

```
9910g (+9868g):
    links:[99480]
    rechts:[99511, 11]
9920g (+9878g):
    links:[99480]
    rechts:[99511, 11]
9930g (+9888g):
    links:[99480]
    rechts:[99511, 11]
9940g (+9898g):
    links:[99480]
    rechts:[99511, 11]
9950g (+9908g):
    links:[99480]
    rechts:[99511, 11]
9960g (+9918g):
    links:[99480]
    rechts:[99511, 11]
9970g (+9928g):
    links:[99480]
    rechts:[99511, 11]
9980g (+9938g):
    links:[99480]
    rechts:[99511, 11]
9990g (+9948g):
    links:[99480]
    rechts:[99511, 11]
10000g (+9958g):
    links:[99480]
    rechts:[99511, 11]
```

Dieser Satz scheint gänzlich ungeeignet für den Bereich zwischen 10 und 10.000 Gramm.

Quelltext

```
class Scale:
    """Ein Szenario für eine Marktwaage"""

    def __init__(self, path: str):
        # Initialisierung aller Attribute
        self.weights: dict[int, int] = {}
        self._possible_weights = {weight: None for weight in range(10, 10010,
10)}
        with open(path, "r") as file:
            for line in range(int(file.readline())):
                weight, amount = file.readline().split(" ")
                self.weights[int(weight)] = int(amount)

    def __repr__(self) -> str:
        return "Scale(" + str(self.weights) + ")"

    def __str__(self) -> str:
        output = ""
        for index, item in enumerate(self.possible_weights.items()):
            weight, combination = item
            if combination[0] > 0:
                combination[0] = "+" + str(combination[0])
```

```
            if combination[0]:
                output += "{}g ({}g):\n    links:{}\n    rechts:
{}\n".format(weight, combination[0],

combination[1], combination[2])
            else:
                output += "{}g:\n    links:{}\n    rechts:{}\n".format(weight,
combination[1], combination[2])
        return output

    @property
    def possible_weights(self) -> dict[int, list[int, list[int], list[int]]]:
        """Die möglichen Kombinationen für jedes Gewicht zwischen 10g und 10kg,
in Schritten von 10g. Jede Seite der
        Waage wird durch eine Liste dargestellt, die die erforderlichen Gewichte
enthält.
        :returns: `dict` mit den Lösungen für jedes Gewicht
        """
        for weight in self._possible_weights:
            self._possible_weights[weight] = self.find_combination(weight)
        return self._possible_weights

    @staticmethod
    def remove_weight(weights: dict[int, int], to_remove: int) -> None:
        """Entfernt das übergebene Gewicht `to_remove` aus der Sammlung der
Gewichte `weights`
        :param weights: Ansammlung von Gewichten als `dict`
        :param to_remove: Gewicht, das aus `weights` entfernt werden soll
        """
        if to_remove not in weights.keys():
            raise ValueError
        weights[to_remove] -= 1
        if not weights[to_remove]:
            del weights[to_remove]

    def find_combination(self, weight: int, weights_copy: dict[int, int] = None,
                         current_combination: list[int, list[int], list[int]] =
None) -> list[
                         int, list[int], list[int]]:
        """Findet rekursiv eine Kombination an Gewichten, die so nah wie möglich
am übergebenen Gewicht
        `weight` dran liegt
        :param weight: gesuchtes Gewicht
        :param weights_copy: `dict` mit Gewichten, mit denen gearbeitet wird
        :param current_combination: Mitlaufwert für Rekursion
        :return: Eine Liste, die die Differenz zum übergebenen Gewicht und die
Gewichte rechts und links enthält"""
        if not current_combination:
            current_combination = [None, [], []]
        # Falls `weight` negativ ist, werden die Seiten der Waage vertauscht
        if weight > 0:
            weight_decrease = current_combination[1]
            weight_increase = current_combination[2]
            weight_factor = 1
        else:
            weight_decrease = current_combination[2]
            weight_increase = current_combination[1]
            weight *= -1
```

```
            weight_factor = -1
        # Falls es noch kein `dict` mit Gewichten gibt, kopiere das des Objekts
        if weights_copy is None:
            weights_copy = self.weights.copy()
        # Gibt es keine Gewichte mehr, kann keine Lösung gefunden werden
        elif not weights_copy:
            current_combination[0] = weight * weight_factor
            return current_combination
        # Ist das gesuchte Gewicht darstellbar als ein einziges Gewichtsstück,
kann dieses als Lösung zurückgegeben
        # werden
        if weight in weights_copy.keys():
            current_combination[0] = 0
            weight_increase.append(weight)
            return current_combination
        # Ist das nicht der Fall, werden alle Kombinationen der vorhandenen
Gewichte durchgespielt und die Kombination
        # mit dem geringsten Abstand zum gesuchten Gewicht wird in
`difference_combination` gespeichert
        min_difference: int = None
        difference_combination: list
        for right_weight in weights_copy.keys():
            for left_weight in weights_copy.keys():
                if left_weight == right_weight:
                    continue
                weight_difference = right_weight-left_weight
                # Entspricht eine Kombination exakt dem gesuchten Gewicht, kann
diese bereits früher als Lösung
                # zurückgegeben werden
                if weight_difference == weight:
                    current_combination[0] = 0
                    weight_decrease.append(left_weight)
                    weight_increase.append(right_weight)
                    return current_combination
                if not min_difference or abs(weight - weight_difference) <
abs(weight - min_difference):
                    min_difference = weight_difference
                    difference_combination = [left_weight, right_weight]
        # Findet das größte einzelne vorhandene Gewicht, das kleiner als das
gesuchte ist
        weight_keys = [key for key in weights_copy]
        weight_keys_index = bisect_left(weight_keys, weight)
        # Je nachdem, ob das einzelne Gewicht oder die Kombination aus zweien
näher am gesuchten Gewicht liegen,
        # wird das eine oder das andere gewählt, um fortzufahren
        if weight_keys_index and min_difference and abs(weight-
weight_keys[weight_keys_index - 1]) <=\
                abs(weight-min_difference) or weight_keys_index and not
min_difference:
            max_weight = weight_keys[weight_keys_index - 1]
            self.remove_weight(weights_copy, max_weight)
            weight_increase.append(max_weight)
            remaining_weight = weight - max_weight
        elif min_difference and abs(weight-min_difference) < weight:
            self.remove_weight(weights_copy, difference_combination[0])
            self.remove_weight(weights_copy, difference_combination[1])
            weight_decrease.append(difference_combination[0])
            weight_increase.append(difference_combination[1])
```

```
            remaining_weight = weight - min_difference
        # Nähert keiner der beiden Ansätze sich der Lösung, kann keine bessere
Lösung gefunden werden
        else:
            current_combination[0] = weight * weight_factor
            return current_combination
        # Rekursion, falls noch Gewicht verbleibt
        return self.find_combination(remaining_weight, weights_copy,
current_combination)
```