Aufgabe 5: Widerstand

Richard Wohlbold
Team-ID: 00012

7. September 2018

Inhaltsverzeichnis

1 Lösungsidee

1.1 Definitionen

- Ein **Elementarwiderstand** ist ein physischer Widerstand aus Zelda's Grabbelkiste
- Ein **zusammengesetzter Widerstand** ist ein Widerstand, der aus einem oder mehreren Elementarwiderständen besteht, die beliebig in Reihe oder parallel geschalten sein können

1.2 Formeln

Zur Bestimmung von R_2, falls R_1 bekannt und R_{ges} aus R_1 und R_2 zusammengebaut ist:

In Reihenschaltung:

$$R_2 = R_{ges} - R_1$$

In Parallelschaltung:

$$R_2 = \frac{1}{\frac{1}{R_{ges}} - \frac{1}{R_1}}$$

1.3 Verfahren

Wenn das Problem für alle $n \in \mathbb{N}$ gelöst werden soll, so müssten alle Kombinationen von Widerständen durchprobiert werden. Da aber ein zusammengesetzten Widerstand aus maximal $n = 4$ Elementarwiderstände bestehen kann, kann ein rekursives Verfahren mit Fallunterscheidung beschrieben werden:

1. Ein Elementarwiderstand:

   Die Lösung ist trivial: Der Elementarwiderstand mit der Ohmzahl, die am nächsten an der geforderten Ohmzahl liegt, wird gewählt.

2. Zwei Elementarwiderstände:

   Zwei Elementarwiderstände können parallel oder in Reihe geschalten werden. Für alle möglichen Widerstände kann ein anderer optimaler Elementarwiderstand R_2, der nach den Formeln oben berechnet wurde, mit dem obigen Verfahren gesucht werden. Beide Möglichkeiten werden einer Menge hinzugefügt. Am Ende wird die Möglichkeit, die am nächsten an der geforderten Ohmzahl R_{ges} liegt, gewählt.

3. Drei Elementarwiderstände:

   Ähnlich wie bei zwei Elementarwiderständen, gibt für jeden Widerstand R_1 aus der Grabbelkiste zwei Möglichkeiten: Er kann in Reihe oder parallel zu einem zusammengesetzten Widerstand R_2 geschalten werden; dabei werden die Formeln oben zur Bestimmung des Widerstandswertes benutzt. Es gibt keine Anordnung von drei Elementarwiderständen, für die R_1 und R_2 keine Elementarwiderstände sind, da jeder zusammengesetzter Widerstand aus mindestens zwei Elementarwiderständen bestehen muss, es aber nur drei gibt. Somit lässt sich jede dieser Anordnungen auf eine, bei der R_1 ein Elementarwiderstand ist, reduzieren.

4. Vier Elementarwiderstände:

   Hier muss zwischen zwei Fällen unterschieden werden:

   a) Anordnungen, bei denen R_1 ein Elementarwiderstand ist, oder die auf einen solchen reduziert werden können. Für solche Anordnungen kann dasselbe Verfahren wie für drei Elementarwiderstände benutzt werden, nur dass hier R_2 aus drei Elementarwiderständen besteht.

   b) Anordnungen, bei denen R_1 kein Elementarwiderstand ist und die nicht auf einen solchen reduziert werden können. Konkret sind dies:

      i. Zwei Widerstandspaare, die ihre Elementarwiderstände parallel schalten, jedoch untereinander in Reihe geschalten sind

      ii. Zwei Widerstandspaare, die ihre Elementarwiderstände in Reihe schalten, jedoch untereinander parallel geschalten sind

Um diese Anordnungen zu berücksichtigen, werden alle Kombinationen von zwei Widerständen in der Kiste bestimmt. Der Liste der möglichen Anordnungen werden somit zwei hinzugefügt: Eine, bei der die zwei Widerstände (R_1) parallel geschalten sind und bei der R_1 mit R_2 in Reihe geschalten ist und eine, bei der die zwei Widerstände (R_1) in Reihe geschalten sind und bei der R_1 mit R_2 parallel geschalten sind. Um R_2 zu ermitteln, kann das Verfahren für zwei Widerstände oben benutzt werden.

Beide Verfahren zusammen stellen sicher, dass alle Anordnungen überprüft werden.

2 Umsetzung

Das verfahren zur Bestimmung eines optimalen zusammengesetzten Widerstands wurde in Python umgesetzt. Das Zeichnen eines Bauplans wird mit dem Python-Paket *graphviz* umgesetzt. Neben dem Paket muss auch das gleichnamige Programm installiert werden, um eine Bildausgabe zu ermöglichen. Die verschiedenen Zeichnungen werden als `.png`-Dateien gespeichert.

Das Skript wird mit zwei Argumenten aufgerufen:

- Das erste Argument ist der Dateiname der Widerstandsliste.
- Das zweite Argument ist der Widerstandswert, der zusammengebaut werden soll.

Ein zusammengebauter Widerstand wird als Tupel repräsentiert: Das erste Feld ist der Typ des zusammengesetzten Widerstands: `einer`, `reihe` oder `parallel`. Die anderen Felder sind abhängig vom Typ:

- Falls der Widerstandstyp `einer` ist, so ist das zweite Feld die Ohmzahl.
- Falls der Widerstandstyp `reihe` ist, so sind das zweite und dritte Feld die in Reihe geschaltenen Widerstände (elementar oder zusammengesetzt).
- Falls der Widerstandstyp `parallel` ist, so sind das zweite und dritte Feld die parallel geschaltenen Widerstände (elementar oder zusammengesetzt).

Die Hauptfunktion, um einen zusammengesetzten Widerstand zu bauen, heißt `widerstand_zusammenbauen`. Beim Aufruf bekommt sie eine Ohmzahl, die der Widerstand haben sollte (`ohm`), eine Zahl, wie viele Elementarwiderstände zu verbauen sind (`wie_viele`), und eine Liste an Widerständen in der Grabbelkiste (`widerstaende`). Sie gibt einen zusammengesetzten Widerstand nach obiger Form zurück. Sie ist rekursiv implementiert:

- Der Basisfall tritt auf, wenn `widerstaende` $= 1$. In diesem Fall wird der Elementarwiderstand mit dem kleinsten Abstand zum geforderten Widerstandswert zurückgegeben.
- Für $n \in \{2, 3\}$ erstellt `wiederstand_zusammenbauen` eine Liste mit Möglichkeiten, indem jeder Widerstand mit einem zusammengesetzten Widerstand aus (`wie_viele` -1) parallel und in Reihe geschalten wird. Damit das Programm schneller läuft falls es Widerstände mehrfach in der Kiste gibt, wird jeder Widerstand hier nur einmal probiert.
- Für $n = 4$ wird neben dem Verfahren für $n \in \{2, 3\}$ zusätzlich ein spezielles Verfahren ausgeführt: Mithilfe von `itertools.combinations` werden alle unterschiedlichen Paare von Widerständen gefunden. Für jede dieser Kombinationen werden zwei Möglichkeiten nach dem Verfahren, das oben beschrieben wurde, der Liste hinzugefügt.

Am Ende von `widerstand_zusammenbauen` wird die Möglichkeit mit der geringsten Differenz zum geforderten Widerstandswert gewählt.

Die Funktion `widerstandswert_berechnen` berechnet den Widerstandswert eines zusammengesetzten Widerstands.

Die Funktion `widerstand_zeichnen` zeichnet einen zusammengesetzten Widerstand auf einem Graphen mithilfe der Bibliothek `graphviz`. `graphviz` muss jeden Knoten eines Graphs identifizieren können. Dazu benutze ich das integrierte Python-Modul `uuid` um IDs für die Knoten des Graphs zu generieren. Die Funktion `widerstand_zeichnen` zeichnet immer zwischen zwei Knoten (`knoten_davor` und `knoten_danach`):

- Bei einem Elementarwiderstand (`einer`) wird einfach ein neuer Knoten mit dem Widerstandswert erstellt und zwei Kanten zu `knoten_davor` und `knoten_danach` erstellt.

- Bei einem zusammengesetzten Widerstand, der zwei Widerstände parallel schaltet, wird `widerstand_zeichnen` für beide aufgerufen. Hierbei bleiben `knoten_davor` und `knoten_danach` gleich.
- Bei einem zusammengesetzten Widerstand, der zwei Widerstände in Reihe schaltet, wird ein neuer Knoten ohne Text hinzugefügt, der dann verwendet werden kann, um die beiden Widerstände zu trennen. So wird `widerstand_zeichnen` zwischen `knoten_davor` und dem neuen Knoten und zwischen `knoten_danach` und dem neuen Knoten aufgerufen.

Um einen Widerstand tatsächlich zu zeichnen, muss erst ein Graph erstellt werden, der mit den ersten beiden Knoten (- und +) befüllt wird. Nachdem alles gezeichnet wurde, wird die Zeichnung als *`n` Widerstände - `rges`.gv.png* abgespeichert, wobei `n` die Anzahl der Widerstände in der Schaltung und `rges` der entstandene Widerstand ist. Außerdem wird die entsprechende `graphviz`-Datei abgespeichert.

3 Beispiele

Bemerkung: Die erste und letzte Zeile der Beispiele sind nur zur Anschaulichkeit übernommen worden. Sie sind nicht Teil der Programmausgabe.

3.1 500 Ω

500 Ω mit der bwinf-Widerstandsliste:

```
$ python Implementierung/widerstand.py beispieldaten/widerstaende.txt 500

1 Widerstand: 470 Ohm
('einer', 470)
Zeichnung nach "1 Widerstände - 470.00 Ohm.gv.png" ausgegeben

2 Widerstände: 500.3802281368822 Ohm
('parallel', ('einer', 560), ('einer', 4700))
Zeichnung nach "2 Widerstände - 500.38 Ohm.gv.png" ausgegeben

3 Widerstände: 500 Ohm
('reihe', ('einer', 180), ('reihe', ('einer', 220), ('einer', 100)))
Zeichnung nach "3 Widerstände - 500.00 Ohm.gv.png" ausgegeben

4 Widerstände: 500.0 Ohm
('parallel', ('reihe', ('einer', 1800), ('einer', 1200)), ('reihe', ('einer', 270), ('einer', 330)))
Zeichnung nach "4 Widerstände - 500.00 Ohm.gv.png" ausgegeben

$
```

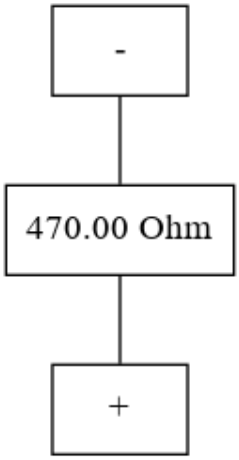

Abbildung 1: 500Ω - 1 Widerstand (470Ω)

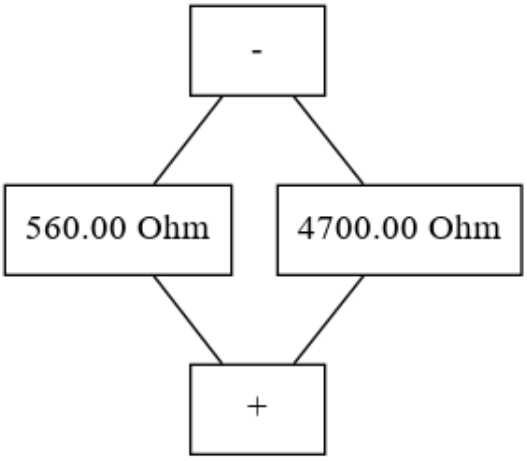

Abbildung 2: 500Ω - 2 Widerstände (500.38Ω)

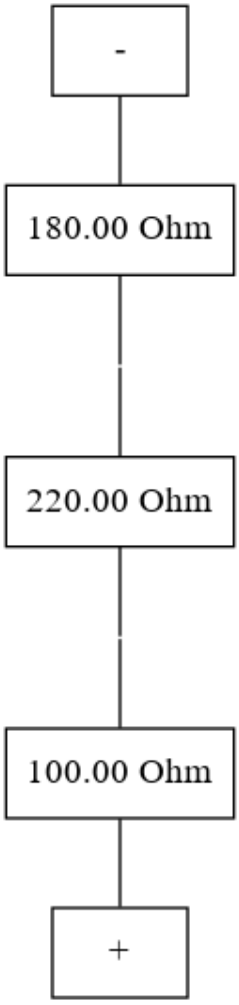

Abbildung 3: 500Ω - 3 Widerstände (500Ω)

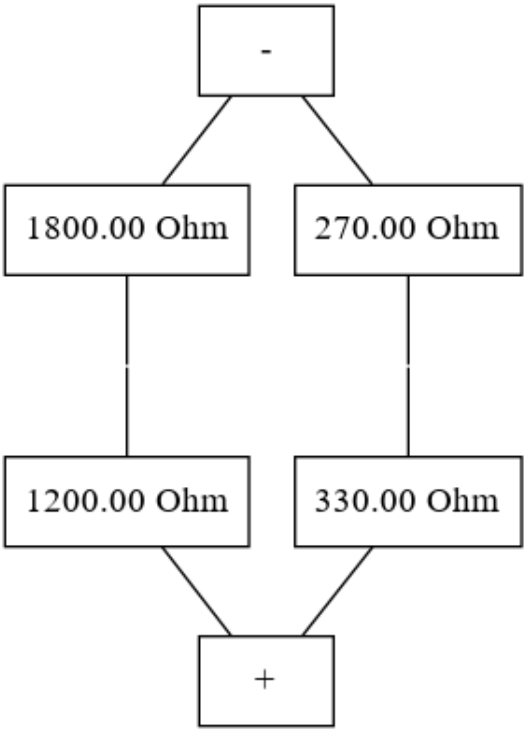

Abbildung 4: 500Ω - 4 Widerstände (500Ω)

3.2 1400 Ω

1400 Ω mit der bwinf-Widerstandsliste:

```
$ python Implementierung/widerstand.py beispieldaten/widerstaende.txt 1400

1 Widerstand: 1500 Ohm
('einer', 1500)
Zeichnung nach "1 Widerstände - 1500.00 Ohm.gv.png" ausgegeben

2 Widerstände: 1406.5573770491803 Ohm
('parallel', ('einer', 2200), ('einer', 3900))
Zeichnung nach "2 Widerstände - 1406.56 Ohm.gv.png" ausgegeben

3 Widerstände: 1400 Ohm
('reihe', ('einer', 390), ('reihe', ('einer', 680), ('einer', 330)))
Zeichnung nach "3 Widerstände - 1400.00 Ohm.gv.png" ausgegeben

4 Widerstände: 1400.0 Ohm
('reihe', ('parallel', ('einer', 1800), ('einer', 2700)),
  ('reihe', ('einer', 220), ('einer', 100)))
Zeichnung nach "4 Widerstände - 1400.00 Ohm.gv.png" ausgegeben

$
```

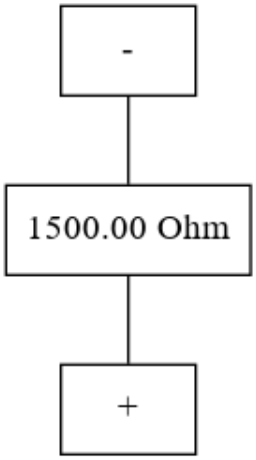

Abbildung 5: 1400Ω - 1 Widerstand (1500Ω)

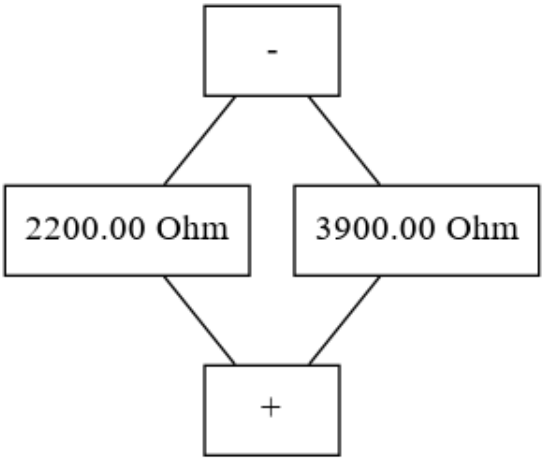

Abbildung 6: 1400Ω - 2 Widerstände (1406.56Ω)

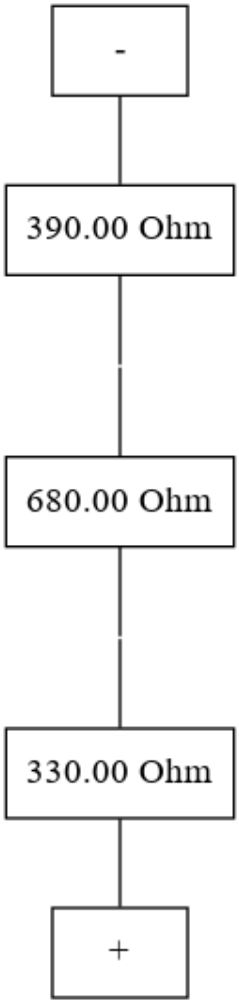

Abbildung 7: 1400Ω - 3 Widerstände (1400Ω)

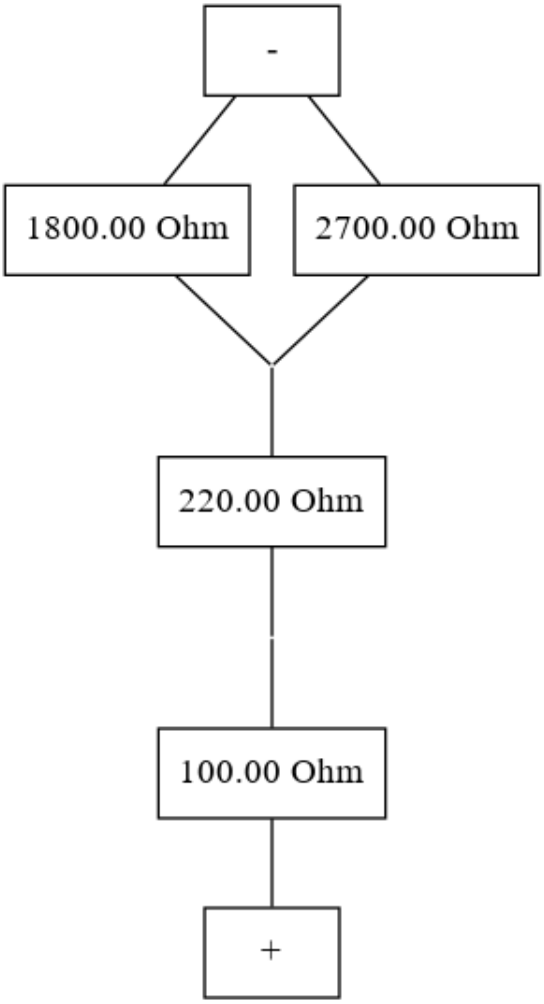

Abbildung 8: 1400Ω - 4 Widerstände (1400Ω)

4 Quellcode (ausschnittsweise)

4.1 widerstand_zusammenbauen

```
def widerstand_zusammenbauen(ohm, wie_viele, widerstaende):
    """
    Einen Widerstand mit einem speziellem Wert zusammenbauen
    @param ohm: Wie viel Ohm der Widerstand am besten haben sollte
    @param wie_viele: Aus wie vielen Elementarwiderstaenden der Widerstand bestehen soll
    @param widerstaende: Liste von zu verbauenden Elementarwiderstaenden
    """
    assert 1 <= wie_viele <= 4

    if wie_viele == 1:
        return ("einer", min(widerstaende, key=lambda x: abs(x-ohm)))

    moeglichkeiten = []

    # Falls noch vier Widerstaende uebrig sind, muessen wir zwei Extrafaelle betrachten
    if wie_viele == 4:
        combs = list(combinations(widerstaende, 2))
        for combination in combs:
            widerstaende.remove(combination[0])
            widerstaende.remove(combination[1])

            # Fall 1: Die Gruppen werden in Reihe geschalten, R1 ist aber parallel geschalten
            uebrig = ohm - 1 / (1 / combination[0] + 1 / combination[1])
```

```
            moeglichkeiten.append(("reihe", ("parallel", ("einer", combination[0]), (
                "einer", combination[1]))), widerstand_zusammenbauen(uebrig, 2, widerstaende)))

            # Fall 2: Die Gruppen werden parallel geschalten, R1 ist aber in Reihe geschalten
            try:
                uebrig = 1 / (1 / ohm - 1 / (combination[0] + combination[1]))
            except ZeroDivisionError:
                # Wenn in der ersten Klammer 0 rauskommt, tritt ein Fehler auf.
                # In diesem Fall ist die Ohmzahl durch die beiden Widerstände gedeckt und die anderen
                # Widerstände sollten so hoch wie möglich sein
                uebrig = 1e100

            moeglichkeiten.append(("parallel", ("reihe", ("einer", combination[0]), (
                "einer", combination[1]))), widerstand_zusammenbauen(uebrig, 2, widerstaende)))


            widerstaende.append(combination[0])
            widerstaende.append(combination[1])

    # Fuer jeden Widerstand zweri Moeglichkeiten erstellen: In Reihe und parallel
    for r1 in set(widerstaende):
        widerstaende.remove(r1)

        # Man kann den neuen Widerstand und den Rest in Reihe schalten
        rest_reihe = ohm - r1
        moeglichkeiten.append(("reihe", ("einer", r1), widerstand_zusammenbauen(
            (rest_reihe), wie_viele-1, widerstaende)))

        # Mann kann den neuen Widerstand und den Rest parallel schalten
        try:
            rest_parallel = 1 / ((1 / ohm) - (1 / r1))
        except ZeroDivisionError:
            # Wenn in der ersten Klammer 0 rauskommt, tritt ein Fehler auf.
            # In diesem Fall ist die Ohmzahl durch r1 gedeckt und die anderen
            # Widerstände sollten so hoch wie möglich sein
            rest_parallel = 1e100

        moeglichkeiten.append(("parallel", ("einer", r1), widerstand_zusammenbauen(
            (rest_parallel), wie_viele-1, widerstaende)))

        widerstaende.append(r1)

    # Die Moeglichkeit mit der hoechsten Genauigkeit finden
    return min(moeglichkeiten, key=lambda x: abs(widerstandswert_berechnen(x)-ohm))
```

code/widerstand_zusammenbauen.py

4.2 widerstand_zeichnen

```
try:
    graphviz = __import__("graphviz")
except ImportError as e:
    print(e)
    print("Fehler beim Versuch, Graphviz zu importieren.
        Es werden keine Zeichnungen ausgegeben.")
    graphviz = None

def widerstand_zeichnen(widerstand, graph, knoten_davor, knoten_danach):
    """
    Den Graphen mit der Bibliothek "graphviz" zeichnen
    Jeder Knoten bekommt eine zufällige ID.
    Alles wird zwischen einen Anfangs- und Endwiderstand gezeichnet.
    Die Funktion wird rekursiv aufgerufen.
    """
    if widerstand[0] == "einer":
        uid = str(uuid4())
        graph.node(uid, label="%.2f Ohm" % widerstand[1])
        graph.edge(knoten_davor, uid)
        graph.edge(uid, knoten_danach)
    elif widerstand[0] == "reihe":
        uid = str(uuid4())
        graph.node(uid, label='', height='0', shape='none')
        widerstand_zeichnen(widerstand[1], graph, knoten_davor, uid)
```

```
25          widerstand_zeichnen(widerstand[2], graph, uid, knoten_danach)
        elif widerstand[0] == "parallel":
27          widerstand_zeichnen(widerstand[1], graph, knoten_davor, knoten_danach)
            widerstand_zeichnen(widerstand[2], graph, knoten_davor, knoten_danach)
29

31  if graphviz is not None:
        name = "%s %s - %.2f Ohm" % (anzahl, w, widerstandswert_berechnen(resultat))
33      graph = graphviz.Graph(name=name, node_attr={
                                'shape': 'box'}, format='png')
35
        graph.node('-', label='-')
37      graph.node('+', label='+')
        widerstand_zeichnen(resultat, graph, '-', '+')
39      graph.render()
        print("Zeichnung nach \"%s.gv.png\" ausgegeben" % name)
```

code/widerstand_zeichnen.py

4.3 widerstandswert_berechnen

```
    def widerstandswert_berechnen(widerstand):
2       """
        Den Widerstandswert eines Widerstandes zurueckgeben
4       @param widerstand: Der Widerstand, fuer den der Widerstandswert berechnet werden soll
        """
6       if widerstand[0] == "einer":
            return widerstand[1]
8       elif widerstand[0] == "reihe":
            return widerstandswert_berechnen(widerstand[1]) +
10              widerstandswert_berechnen(widerstand[2])
        elif widerstand[0] == "parallel":
12          return 1 / (1 / widerstandswert_berechnen(widerstand[1]) +
                1 / widerstandswert_berechnen(widerstand[2]))
14      print("error")
        exit(1)
```

code/widerstandswert_berechnen.py