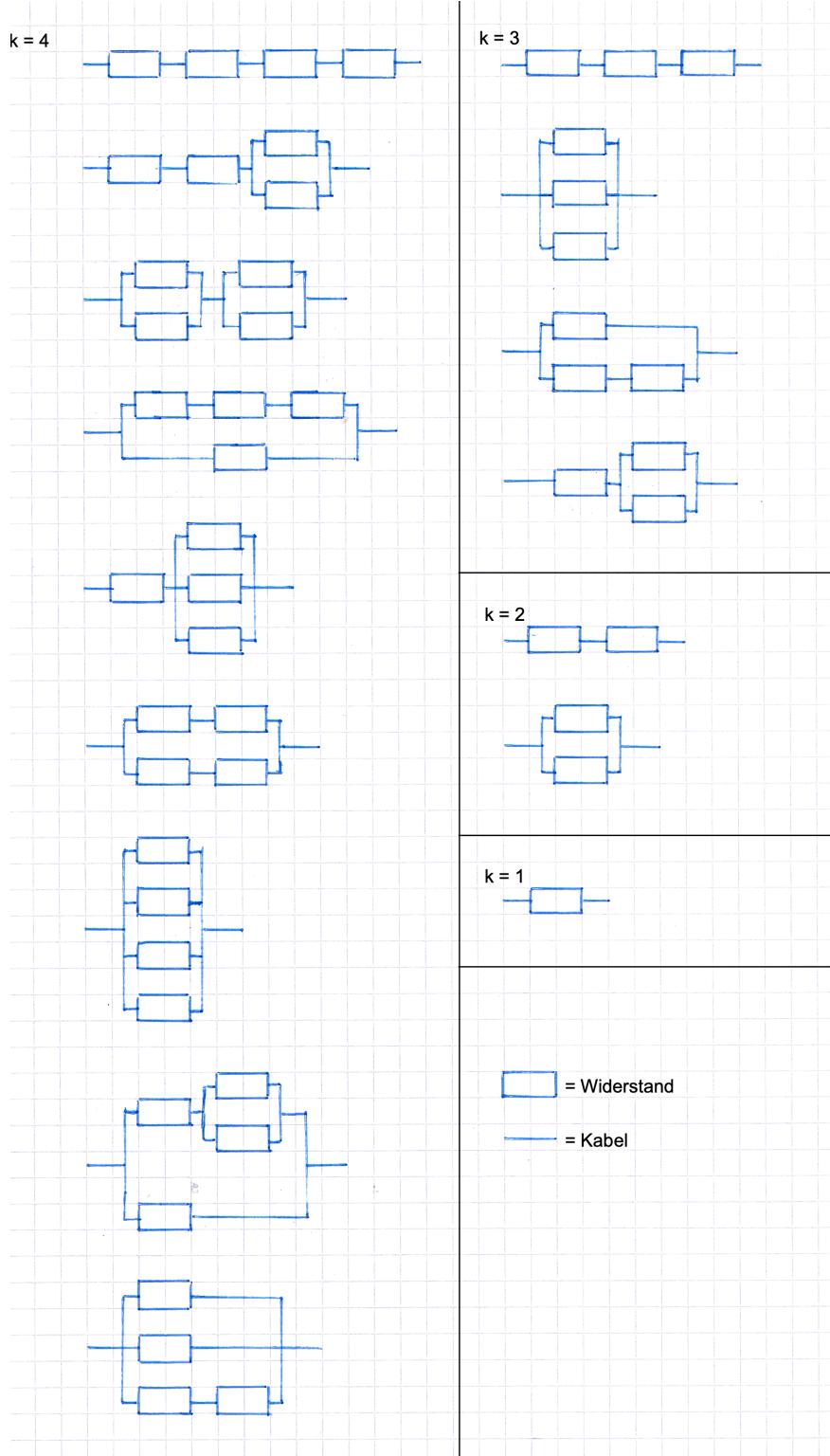


Dokumentation zur Aufgabe „Widerstand“

Die Lösungsidee:

Da es bei dieser Aufgabe eine festgelegte Anzahl an maximal zu verwendenden Widerständen gibt, muss man lediglich alle möglichen Baupläne ausprobieren. Dabei muss beachtet werden, dass alle Bauteile stets nur einmal in einem Bauplan verwendet werden können.

Am Ende sollten zudem sämtliche Baupläne, die das möglichst beste Ergebnis erzielen, ausgegeben werden, da man so eventuell unterschiedliche Teile am Ende übrig hat, welche in manchen Fällen dann vom Nutzer für andere benötigte Widerstände benutzt werden können. Die möglichen Baupläne, die sich mit bis zu vier Widerständen bauen lassen, die ich selber gefunden habe, sind die folgenden:



Die Umsetzung mit Quelltext:

Die Umsetzung der Lösungsidee erfolgt in meiner Bearbeitung in der Programmiersprache Ruby. Das Programm beziehungsweise Script („widerstand.rb“) liegt im Ordner „Aufgabe 5“ und stellt zeitgleich Quelltext und ausführbare Datei dar. Ich habe das Script mit der Ruby Version 2.5.3-1-x64 in der CMD von Windows 10 getestet.

```
puts "Gebe den Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B.  
widerstaende.txt)"  
file = gets.chomp  
$resistances = File.open(file).read.force_encoding("UTF-8").split("\n")  
puts "Gebe den gesuchten Widerstandswert in Ω ein: "  
$neededOhm = gets  
puts "Gebe die maximal zu verwendenden Widerstände an (k = 1, ..., 4):"  
k = gets  
$neededOhm = $neededOhm.to_f  
k = k.to_f
```

Mit dem ersten Codeabschnitt werden die Liste aller verfügbaren Widerstände, der gesuchte Widerstandswert und die maximal zu benutzenden Widerstände durch den Nutzer festgelegt.

```
def seriell(r)  
    i = 0  
    newOhm = 0.0  
    while i < r.length do  
        newOhm += r[i]  
        i += 1  
    end  
    return newOhm  
end  
  
def parallel(r)  
    i = 0  
    newOhm = 0.0  
    while i < r.length do  
        newOhm += 1.0 / r[i]  
        i += 1  
    end  
    newOhm = 1.0 / newOhm  
    return newOhm  
end
```

Die Methoden *seriell* und *parallel* bekommen jeweils einen Array übergeben, dessen Zahlen zusammengerechnet und zurückgegeben werden. Bei der Methode *seriell* werden dazu alle Zahlen des übergebenen Arrays addiert. Bei der Methode *parallel* hingegen werden alle Kehrwerte der Zahlen des übergebenen Arrays addiert und anschließend der Kehrwert dieser zusammengerechneten Zahl zurückgegeben. So lässt sich jeder Bauplan mit nur zwei Methoden nachbauen.

Anschließend folgen die vier Methoden $k4$, $k3$, $k2$ und $k1$. Da diese alle auf die selbe Art funktionieren folgt lediglich eine Erklärung an der Methode $k2$:

```
def k2
  a = 0
  b = 0
  k2List1 = Array.new
  k2List2 = Array.new
  k2List = Array.new
  firstResistanceList = Array.new
  secondResistanceList = Array.new
```

$k2List1$ und $k2List2$ beinhalten alle Ergebnisse der Kombinationen von Widerständen. Dabei beinhaltet $k2List1$ alle Ergebnisse durch serielles Verbinden und $k2List2$ alle Ergebnisse durch paralleles Verbinden von Widerständen. Genau so gibt es in den Methoden $k3$ und $k4$ für jeden Bauplan, der aus drei beziehungsweise vier Widerständen besteht einen Array, welcher die Ergebnisse des Verbindens nach dem jeweiligen Bauplan beinhaltet. Auch die Methode $k1$ hat einen Array nach diesem Muster, auch wenn hier letztlich nur geguckt werden muss, welcher der vorhandenen Widerstände am nächsten am gesuchten Widerstand liegt. $k2List$ beinhaltet am Ende die besten Baupläne für den gesuchten Widerstand. Dieser Array wird am Ende der Methode zurückgegeben. a und b sind dazu da, alle möglichen Baupläne durchzugehen. Es gibt für zwei Widerstände $Bauteile * (Bauteile - 1)$ Möglichkeiten, welche der Widerstände verwendet werden. Für drei Bauteile sind es folglich $Bauteile * (Bauteile - 1) * (Bauteile - 2)$ Möglichkeiten. Im nachfolgenden Abschnitt werden jedoch einfach $Bauteile * Bauteile$ Möglichkeiten durchlaufen (für $k = 3$ $Bauteile * Bauteile * Bauteile$ usw.). Dies führt grundsätzlich zu dem Problem, dass Widerstände mehrfach verwendet werden könnten: Es gibt in der Methode $k2$ zwei (in $k3$ folglich drei usw.) ineinander liegende Schleifen, welche jeweils solange laufen, bis a bzw. b (in $k3$ zudem c usw.) bis auf die Anzahl der vorhandenen Widerstände hochgezählt wurde. Die inneren Schleifen werden beim Durchlauf der äußeren Schleifen also immer wieder neu ausgeführt. In der innersten Schleife werden dann die verschiedenen möglichen Baupläne mit dem jeweils a -ten und b -ten Widerstand ausprobiert. Um die doppelte Verwendung eines Widerstandes zu vermeiden, geschieht dies jedoch nur, wenn a nicht gleich b ist (in $k3$ folglich $a \neq b$ und $a \neq c$ und $b \neq c$ usw.). $firstResistanceList$ und $secondResistanceList$ (in $k3$ zudem $thirdResistanceList$ usw.) enthalten zudem die verwendeten Widerstände aus jedem Durchlauf der innersten Schleife:

```
while a < $resistances.length do
  b = 0
  while b < $resistances.length do
    if not a == b
      distance = seriell([$resistances[a].to_f, $resistances[b].to_f]) - $neededOhm
      if distance < 0
        distance = -distance
      end
      k2List1.push(distance)
      distance = parallel([$resistances[a].to_f, $resistances[b].to_f]) - $neededOhm
      if distance < 0
        distance = -distance
      end
      k2List2.push(distance)

      firstResistanceList.push(a)
      secondResistanceList.push(b)
    end
    b += 1
  end
  a += 1
end
```

```

minInList = k2List1.min
while not k2List1.index(minInList) == nil do
    indexOfMin = k2List1.index(minInList)
    k2List.push([firstResistanceList[indexOfMin], secondResistanceList[indexOfMin], 1,
k2List1.min])
    k2List1[indexOfMin] = minInList + 1
end

minInList = k2List2.min
while not k2List2.index(minInList) == nil do
    indexOfMin = k2List2.index(minInList)
    k2List.push([firstResistanceList[indexOfMin], secondResistanceList[indexOfMin], 2,
k2List2.min])
    k2List2[indexOfMin] = minInList + 1
end

return k2List
end

```

Zum Schluss werden dann die jeweils besten erreichten Widerstandswerte eines jeden Bauplans in einem Array in dem Array *k2List* (bzw. *k3List* usw.) gespeichert. In jedem Array innerhalb des Arrays *k2List* befinden sich nach einander folgende Einträge: erstes Bauteil, zweites Bauteil, (drittes Bauteil, viertes Bauteil,) Bauplan in Form eines Integers, welcher später in der Ausgabe zum ausgeben des Bauplans benutzt wird, sowie der Abstand des so erreichten Widerstandes zum gesuchten Widerstand.

```

outputList = Array.new
ListForK1 = k1
min = ListForK1[0][2].to_f + 1.0

i = 0
while i < ListForK1.length do
    if ListForK1[i][2].to_f < min
        min = ListForK1[i][2].to_f
        outputList.clear
        outputList.push(ListForK1[i])
    elsif ListForK1[i][2].to_f == min
        outputList.push(ListForK1[i])
    end
    i += 1
end

```

Der Array *outputList* beinhaltet die besten Baupläne, die gefunden wurden. *ListForK1* ruft die Methode *k1* auf und beinhaltet somit die Widerstände die am nächsten am gesuchten Widerstand liegen. Wird der Widerstand 330Ω gesucht und es gibt die Widerstände 315Ω und 345Ω , so finden sich beide in *ListForK1* wieder, sollten sie am nächsten an dem gesuchten Wert liegen, da sie gleich weit vom diesem entfernt liegen.

Da es nicht möglich ist ohne Widerstände einem gesuchten Widerstand möglichst nahe zu kommen, kann immer nach einer Lösung für $k = 1$ gesucht werden. Die while Schleife hier durchsucht den gesamten Array *ListForK1* nach den besten Wert(en) und fügt diesen (/ diese) an den Array *outputList* an.

```

if k >= 2
ListForK2 = k2

i = 0
while i < ListForK2.length do
    if ListForK2[i][3].to_f < min
        min = ListForK2[i][3].to_f
        outputList.clear
        outputList.push(ListForK2[i])
    elsif ListForK2[i][3].to_f == min
        outputList.push(ListForK2[i])
    end
    i += 1
end
end

if k >= 3
ListForK3 = k3

i = 0
while i < ListForK3.length do
    if ListForK3[i][4].to_f < min
        min = ListForK3[i][4].to_f
        outputList.clear
        outputList.push(ListForK3[i])
    elsif ListForK3[i][4].to_f == min
        outputList.push(ListForK3[i])
    end
    i += 1
end
end

if k == 4
ListForK4 = k4

i = 0
while i < ListForK4.length do
    if ListForK4[i][5].to_f < min
        min = ListForK4[i][5].to_f
        outputList.clear
        outputList.push(ListForK4[i])
    elsif ListForK4[i][5].to_f == min
        outputList.push(ListForK4[i])
    end
    i += 1
end
end

```

Der selbe Prozess wiederholt sich für $k = 2$, $k = 3$ und $k = 4$, wobei bei $k = 3$ auch die Möglichkeiten aus $k = 2$ und bei $k = 4$ auch die Möglichkeiten aus $k = 2$ und $k = 3$ ausprobiert werden.

Jede der while Schleifen funktioniert, indem jeder Eintrag aus den Listen aus den Methoden $k1$, $k2$, $k3$ und $k4$ kontrolliert wird. Ist ein dabei gefundener Wert näher am gesuchten Wert als der vorherige nächste Wert, wird der Array *outputList* geleert und bekommt diesen Wert stattdessen. Ist ein gefundener Wert gleich nah am gesuchten Widerstand, wird dieser an den Array *outputList* angehängt. Am Ende bleiben so alle besten Baupläne mit dem einheitlich besten Ergebnis über.

```

puts "\nMit den Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von
#${neededOhm} Ω mit einem Abstand von #{min} Ω auf folgende Arten erreichen:"
i = 0
while i < outputList.length do
  if outputList[i].length == 3
    puts $resistances[outputList[i][0]].to_s
  elsif outputList[i].length == 4
    if outputList[i][2] == 1
      puts $resistances[outputList[i][0].to_i].to_s + " seriell " + $resistances[outputList[i][1].to_i].to_s + " = " + seriell([$resistances[outputList[i][0].to_i].to_f, $resistances[outputList[i][1].to_i].to_f]).to_s
    elsif outputList[i][2] == 2
      puts $resistances[outputList[i][0].to_i].to_s + " parallel " + $resistances[outputList[i][1].to_i].to_s + " = " + parallel([$resistances[outputList[i][0].to_i].to_f, $resistances[outputList[i][1].to_i].to_f]).to_s
    end
  elsif outputList[i].length == 5
    if outputList[i][3] == 1
      puts $resistances[outputList[i][0].to_i].to_s + " seriell " + $resistances[outputList[i][1].to_i].to_s + " seriell " + $resistances[outputList[i][2].to_i].to_s + " = " +
      seriell([$resistances[outputList[i][0].to_i].to_f, $resistances[outputList[i][1].to_i].to_f, $resistances[outputList[i][2].to_i].to_f]).to_s
    elsif outputList[i][3] == 2
      puts $resistances[outputList[i][0].to_i].to_s + " seriell (" + $resistances[outputList[i][1].to_i].to_s + " parallel " + $resistances[outputList[i][2].to_i].to_s + ") = " +
      seriell([$resistances[outputList[i][0].to_i].to_f, parallel([$resistances[outputList[i][1].to_i].to_f, $resistances[outputList[i][2].to_i].to_f])]).to_s
    elsif outputList[i][3] == 3
      puts $resistances[outputList[i][0].to_i].to_s + " parallel " + $resistances[outputList[i][1].to_i].to_s + " = " +
      parallel([$resistances[outputList[i][0].to_i].to_f, $resistances[outputList[i][1].to_i].to_f, $resistances[outputList[i][2].to_i].to_f]).to_s
    elsif outputList[i][3] == 4
      puts $resistances[outputList[i][0].to_i].to_s + " parallel (" + $resistances[outputList[i][1].to_i].to_s + " seriell " + $resistances[outputList[i][2].to_i].to_s + ") = " +
      parallel([$resistances[outputList[i][0].to_i].to_f, seriell([$resistances[outputList[i][1].to_i].to_f, $resistances[outputList[i][2].to_i].to_f])]).to_s
    end
  [...]
end

```

Im letzten Abschnitt folgt dann die Ausgabe aller Baupläne im Array *outputList*. Dazu wird jeder Array innerhalb des Arrays *outputList* auf seine Länge kontrolliert. Besteht ein Array zum Beispiel nur aus drei Einträgen, dann gehört er zur $k = 1$ Reihe: erster Eintrag: Position des Widerstands in eingelesener Liste, zweiter Eintrag: der Bauplan (hier: 1) und dritter Eintrag: der Abstand zum gesuchten Widerstand.

Bei einem Array, innerhalb des Arrays *outputList*, mit vier Einträgen handelt es sich folglich um einem Bauplan der $k = 2$ „Baureihe“: erster Eintrag: Position des ersten Widerstands in der eingelesenen Liste, zweiter Eintrag: Position des zweiten Widerstands in der eingelesenen Liste, dritter Eintrag: der Bauplan (hier: 1 oder 2) und vierter Eintrag: der Abstand zum gesuchten Widerstand.

Auf die selbe Art kann man auch Einträge aus der $k = 3$ und $k = 4$ Reihe wiederfinden (Aus Platz gründen hier nicht extra aufgeführt).

Anhand des Integers, der an vorletzter Stelle in den Arrays mitgeliefert wird, kann herausgefunden werden, wie die davor liegenden Widerstände zusammen gebaut sein müssen. Folglich erfolgt für jede Nummer ein dieser Nummer entsprechender Output der Widerstände. Zur Überprüfung werden die aus den Bauplänen folgenden Widerstände noch einmal berechnet und mit ausgegeben.

Beispiel (Demonstration):

```
GW: Eingabeaufforderung

T:\BWINF 2018\Widerstand>widerstand
Geben den Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
widerstaende.txt
Geben den gesuchten Widerstandswert in Ω ein:
314
Gebe die maximal zu verwendenden Widerstände an (k = 1, ..., 4):
4

Mit den Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 314.0 Ω mit einem Abstand von 0.0006572461387008843 Ω auf folgende Arten erreichen:
(120 seriell [470 parallel 1200]) parallel 1000 = 313.9993427538613
(120 seriell [1200 parallel 470]) parallel 1000 = 313.9993427538613

I:\BWINF 2018\Widerstand>widerstand
Geben den Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
widerstaende.txt
Geben den gesuchten Widerstandswert in Ω ein:
314
Gebe die maximal zu verwendenden Widerstände an (k = 1, ..., 4):
3

Mit den Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 314.0 Ω mit einem Abstand von 0.0556368960468496 Ω auf folgende Arten erreichen:
330 parallel (1800 seriell 4700) = 314.05563689604685
330 parallel (4700 seriell 1800) = 314.05563689604685

I:\BWINF 2018\Widerstand>widerstand
Geben den Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
widerstaende.txt
Geben den gesuchten Widerstandswert in Ω ein:
314
Gebe die maximal zu verwendenden Widerstände an (k = 1, ..., 4):
2

Mit den Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 314.0 Ω mit einem Abstand von 0.7265077138849847 Ω auf folgende Arten erreichen:
6800 parallel 330 = 314.726507713885
330 parallel 6800 = 314.726507713885

I:\BWINF 2018\Widerstand>widerstand
Geben den Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
widerstaende.txt
Geben den gesuchten Widerstandswert in Ω ein:
314
Gebe die maximal zu verwendenden Widerstände an (k = 1, ..., 4):
1

Mit den Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 314.0 Ω mit einem Abstand von 16.0 Ω auf folgende Arten erreichen:
330

Gabeaufforderung
en Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
taende.txt
en gesuchten Widerstandswert in Ω ein:
ie maximal zu verwendenden Widerstände an (k = 1, ..., 4):

n Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 315.0 Ω mit einem Abstand von 0.0 Ω auf folgende Arten erreichen:
parallel 560 parallel 1200 = 315.0
parallel 1200 parallel 560 = 315.0
parallel 1800 parallel 1200 = 315.0
parallel 1200 parallel 1800 = 315.0
parallel 560 parallel 560 = 315.0
parallel 560 parallel 1800 = 315.0
parallel (330 seriell 390) = 315.0
parallel (390 seriell 330) = 315.0
parallel (330 seriell 390 parallel 180) = 315.0
parallel (180 parallel 180 parallel 180) = 315.0
parallel (150 parallel 180 parallel 180) = 315.0
parallel (150 parallel 180 parallel 180) = 315.0
parallel (180 parallel 180 parallel 150) = 315.0
parallel (180 parallel 180 parallel 180) = 315.0
parallel (180 seriell 180 parallel 330) = 315.0
parallel (180 seriell 390 seriell 150) = 315.0
parallel (150 seriell 180 seriell 390) = 315.0
parallel (150 seriell 180 seriell 470) = 315.0
parallel (150 seriell 470 seriell 180) = 315.0
parallel (150 seriell 470 seriell 180) = 315.0
parallel (160 seriell 150 seriell 470) = 315.0
parallel (160 seriell 470 seriell 150) = 315.0
parallel (330 seriell 120 seriell 270) = 315.0
parallel (330 seriell 270 seriell 120) = 315.0
parallel (330 seriell 180 seriell 150) = 315.0
parallel (390 seriell 150 seriell 180) = 315.0
parallel (120 seriell 330 seriell 270) = 315.0
parallel (120 seriell 270 seriell 330) = 315.0
parallel (470 seriell 180 seriell 150) = 315.0
parallel (470 seriell 180 seriell 150) = 315.0
parallel (270 seriell 330 seriell 150) = 315.0
parallel (270 seriell 120 seriell 330) = 315.0
parallel (1000 parallel 1800) parallel 960 = 315.0
parallel (1500 parallel 1800) parallel 960 = 315.0

NF 2018\Widerstand>widerstand
en Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
taende.txt
en gesuchten Widerstandswert in Ω ein:
ie maximal zu verwendenden Widerstände an (k = 1, ..., 4):

n Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 315.0 Ω mit einem Abstand von 0.0 Ω auf folgende Arten erreichen:
parallel 560 parallel 1700 = 315.0
parallel 1200 parallel 560 = 315.0
parallel 1200 parallel 1200 = 315.0
parallel 1200 parallel 1800 = 315.0
parallel 560 parallel 560 = 315.0
parallel (330 seriell 390) = 315.0
parallel (390 seriell 330) = 315.0

NF 2018\Widerstand>widerstand
en Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
taende.txt
en gesuchten Widerstandswert in Ω ein:
ie maximal zu verwendenden Widerstände an (k = 1, ..., 4):

n Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 315.0 Ω mit einem Abstand von 0.2734922861150153 Ω auf folgende Arten erreichen:
parallel 330 = 314.726507713885
parallel 6800 = 314.726507713885

NF 2018\Widerstand>widerstand
en Namen einer .txt-Datei mit vorhandenen Widerständen ein (z.B. widerstaende.txt)
taende.txt
en gesuchten Widerstandswert in Ω ein:
ie maximal zu verwendenden Widerstände an (k = 1, ..., 4):

n Widerstandswerten aus der Liste lässt sich der gesuchte Widerstand von 315.0 Ω mit einem Abstand von 15.0 Ω auf folgende Arten erreichen:
```

Zu sehen sind die Ausgaben für einen gesuchten Widerstandswert von 314Ω und 315Ω mit nach einander vier, drei, zwei und einem Bauteilen aus der Datei „widerstaende.txt“.