Aufgabe 5: Widerstand

Team-ID: 00126
Team-Name: GSO
Bearbeiter dieser Aufgabe: Niklas Degener

Inhaltsverzeichnis:

Lösungsidee:

Zuerst wird überprüft, ob der Widerstand bereits in der Grabbelkiste existiert. Falls dies nicht der Fall ist, wird zuerst für jeden Widerstand die Distanz zum Zielwert ausgerechnet und die niedrigste gespeichert.
Dann werden alle Kombination aus zwei Widerständen gespeichert. Falls die Kombination, die Distanz der zweier Kombination, die am nächsten am Zielwert liegt kleiner ist als die Distanz des zuvor gespeicherten Widerstands, wird er durch die zweier Kombination ersetzt.
Mit den Kombination, die aus drei oder vier Widerständen bestehen wird gleich verfahren, sodass der gespeicherte Wert am Ende die Kombination ist, die von allen möglichen am nächsten am Zielwert liegt.

Umsetzung:

Die beschriebene Lösungsidee wurde in der Programmiersprache Java implementiert.

Hierzu wurde eine Klasse “Solution” (siehe S.4) erstellt, die eine Lösung beschreibt. In dieser Klasse sind die Kombinationen von Reihen- und Parallelschaltungen, die verwendet Widerstände, als auch der Wert der kombinierten Widerstände und dessen Distanz zum Zielwert.

In der Klasse “Widerstand”, die die main-Methode enthält, wird in der Konstante “goal” der Zielwert festgelegt und die Widerstände werden mithilfe der “readFile” methode eingelesen.
Dann wird eine Instanz der Klasse “WiderstandCalculator” erstellt.

Im Konstruktor der Klasse wird mit hilfe der “exists” methode überprüft, ob der Widerstand bereits existiert. Ist dies nicht der Fall, wird das Ergebnis der “lowestDistOfExisting”, also der Widerstand, der am nächsten am Zielwert liegt zwischengespeichert.
Danach werden von der “Calc” Methode, die jeweiligen “getLowestValues” Methoden (getLowest2Values, getLowest3Values, getLowest4Values(siehe S.4-5)) aufgerufen und das Ergebnis, falls es kleiner als der vorherige gespeicherte Wert ist in die Variable “finalSol” gespeichert.
Diese “getLowestValues” Methoden verwenden die jeweiligen “combination” Methoden (siehe S.6), um die jeweiligen Widerstände mit Reihen- und Parallelschaltungen zu kombinieren.

In den “combination” Methoden repräsentiert der Wert 0 der “condition” parameter eine Reihen und der Wert 1 eine Parallelschaltung. Die Integer “in” repräsentieren den Index des verwendeten Widerstands im “resistor” array.

Diese “combination” Methoden rufen die “reihe” und “parallel” Methoden(siehe S.5) auf, um Widerstände miteinander zu kombinieren.

Zuletzt wird “finalSol” mit Hilfe der “printSolution” ausgegeben.

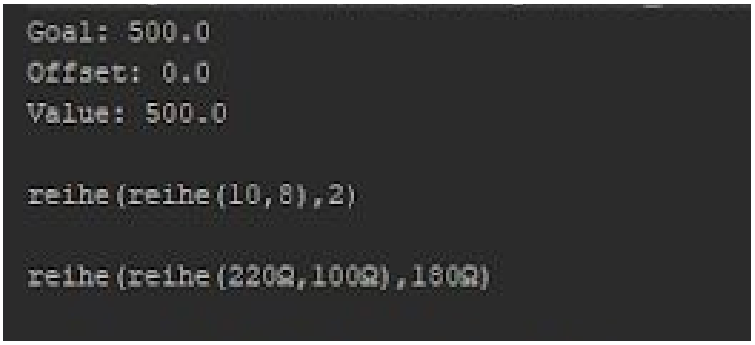
Lösungsbeispiele:

Mögliche Ausgaben des Programms:

Fall 1, zu suchende Widerstand existiert in der Grabbelkiste:

```
The resistor exists
```

Fall 2, zu suchender Widerstand existiert nicht in der Grabbelkiste:



Goal gibt den Zielwert an, Offset die Distanz vom gefundenen Wert zum Zielwert und Value den Wert der gefundenen Kombination aus Widerständen.
Die fünfte Zeile gibt die an, wie die Widerstände, hier als Index im WiderstandsArray angegeben, miteinander kombiniert werden müssen.
Die letzte Zeile der Ausgabe tut dies auch, hier werden aber die genauen Werte der benutzten Widerstände angegeben.

Für die Widerstände, die in „bwinf.de/fileadmin/user_upload/BwInf/2018/37/1._Runde/Material/widerstaende.txt“ angegeben sind, findet das Programm die folgenden Lösungen:

Für Zielwert 500: reihe(reihe(220Ω,100Ω),180Ω) ; Ergebnis: 500Ω
Für Zielwert 140: parallel(reihe(reihe(120Ω,1800Ω),180Ω),150Ω) ; Ergebnis: 140Ω
Für Zielwert 314: parallel(reihe(parallel(1200Ω,470Ω),120Ω),1000Ω); Ergebnis: 313,9993428Ω
Für Zielwert 315: parallel(reihe(390Ω,330Ω),560Ω); Ergebnis: 315Ω
Für Zielwert 1620: reihe(120Ω,1500Ω); Ergebnis: 1620Ω
Für Zielwert 2719: reihe(reihe(parallel(220Ω,180Ω),1800Ω),820Ω); Ergebnis: 2719Ω
Für Zielwert 4242: reihe(reihe(parallel(120Ω,180Ω),270Ω),3900Ω); Ergebnis: 4242Ω

Quelltext

Solution.java

```
public class Solution {
  public double dist;
  public double nearest;
  public int[] conditions;
  public int[] values;
  public Solution(){
    conditions = new int[3];
    values = new int[4];
  }
}
```

WiderstandCalculator.java:getLowest4Values

```
public Solution getLowest4Values(){
   Solution s = new Solution();

   double dist = 0;
   double nearest = 0;
   int[] conditions = {0,0,0};
   int[] values = {0,0,0,0};

   boolean firstOp = true;

   outer: for(int i = 0; i < 2; i++){
       for(int j = 0; j < 2; j++){
           for(int k = 0; k < 2; k++){
               for(int l = 0; l < resistors.length; l++){
                   for(int m = 0; m < resistors.length; m++){
                       for(int n = 0; n < resistors.length; n++){
                           for(int o = 0; o <
resistors.length;o++){
                               if(l!=o && m!=o && n!=o &&
                                l!=m && l!=n && m!=n) {
                                   double value = combination(i,
                                  j, k, resistors[l],
                                 resistors[m],
                                   resistors[n], resistors[o]);
                                    if(firstOp){
                                        firstOp = false;
                                        dist =
Math.abs(value-goal);
                                        nearest = value;

                                        conditions = new int[]{i,j,k};
```

```
                                        values = new
int[]{l,m,n,o};

                                        if(value == goal) break outer;
                                    }else{
                                        if (Math.abs(value - goal)
<
                                       dist){
                                            dist = Math.abs(value -
                                           goal);
                                            nearest = value;

                                            conditions = new
                                           int[]{i,j,k};
                                            values = new
                                           int[]{l,m,n,o};

                                            if(value == goal) break
                                           outer;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    s.dist = dist;
    s.conditions = conditions;
    s.values = values;
    s.nearest = nearest;
    return s;
}
```

WiderstandCalculator.java:parallel

```
public double parallel(double value1, int value2){
    return (1/(((double)1/value1)+((double)1/value2)));
}
```

WiderstandCalculator.java:reihe

```
public double reihe(double value1, int value2){
```

```
    return value1 + value2;
}
```

WiderstandCalculator.java:combination

```
public double combination(int condition1, int condition2, int
condition3, int in1, int in2, int in3, int in4){
    double firstVal;
    double secondVal;
    double thirdVal;
    if(condition3 == 1)
    {
        firstVal = parallel(in4, in3);
    }else{
        firstVal = reihe(in4, in3);
    }
    if(condition2 == 1){
        secondVal = parallel(firstVal, in2);
    }else{
        secondVal = reihe(firstVal, in2);
    }
    if(condition1 == 1){
        thirdVal = parallel(secondVal, in1);
    }else{
        thirdVal = reihe(secondVal, in1);
    }
    return thirdVal;
}
```