Aufgabe 5: Widerstände

Team: Jan Niklas Groeneveld
Team-ID: 00828

22. November 2018

Inhaltsverzeichnis

1 Lösungsidee **1**

2 Umsetzung **2**

3 Beispiele **4**

4 Quellcode **6**

1 Lösungsidee

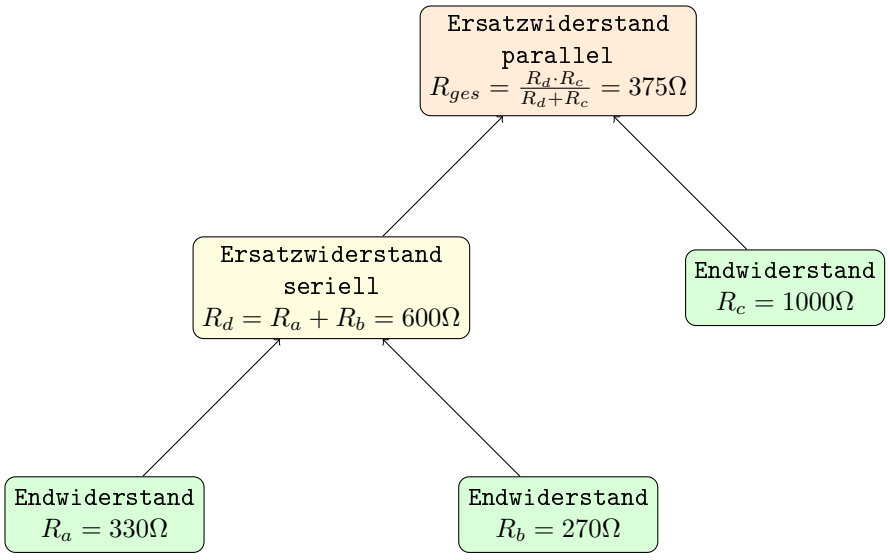
Als Grundidee liegt dem Algorithmus zu Grunde, dass ein Widerstand in einer Schaltung immer durch zwei ersetzt werden kann. Für eine serielle Verschaltung gilt:

$$R_E = R_1 + R_2 \tag{1}$$

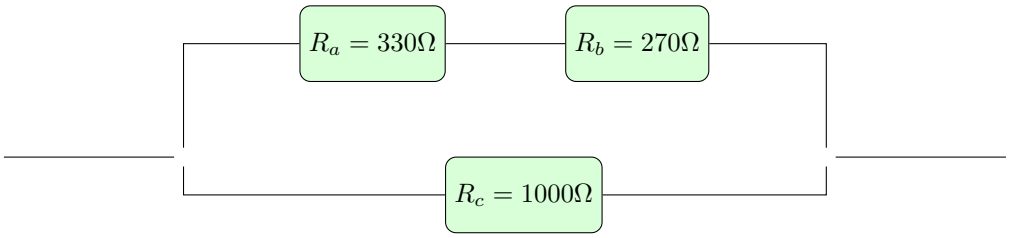
Bei einer Parallelschaltung wird hingegen über den Kehrwert addiert:

$$\frac{1}{R_E} = \frac{1}{R_1} + \frac{1}{R_2} \Rightarrow R_E = \frac{R_1 \cdot R_2}{R_1 + R_2} \tag{2}$$

Das Programm soll deswegen in einer Baumstruktur einen Widerstand durch zwei ersetzen, und nach anschließender Kombination der Widerstandswerte aus der sogenannten Grabbelkiste für die Ersatzwiderstaende regelkonform den entsprechenden Widerstandswert berechnen. In der folgenden Grafik ist dargestellt, wie ein Widerstand mit 375Ω durch Kombination dreier Widerstände ersetzt wird:



Als Schaltplan sähe die Schaltung so aus:



Um das zu erreichen, erstellt das Programm zuerst Baupläne, indem es, von einem Startpunkt ausgehend, so viele Widerstände ersetzt, dass die geforderte Zahl von Endwiderständen (zwei bis vier in der Aufgabe) mit Werten versehen werden können. Dann werden für jeden Bauplan alle Möglichkeiten der Kombination der Widerstandswerte ausprobiert. Weil bei einem Umfang von 24 Widerständen jeder nur einmal verwendet werden darf, ergeben sich bei vier zu verwendenden Widerständen für jeden Bauplan $n = \frac{24!}{20!} = 255024$ Kombinationsmöglichkeiten. Bei zwölf durchzurechnenden Bauplänen für vier Widerstände müssen insgesamt also maximal 3.060.288 Möglichkeiten durchgerechnet werden. Das wird von einem Computer in sehr kurzer Zeit erledigt. Wenn jedoch der Betrag der Differenz aus erreichtem und angestrebten Wert, derjenige Wert, der zu minimieren ist, vorher auf genau null fällt, muss nicht weiter gerechnet werden, da bereits eine optimale Lösung gefunden wurde.

Es muss jedoch beachtet werden, dass eine Kombination von vier Widerständen im Bastelgebrauch nicht von Relevanz ist, da Bastelwiderstände eine Toleranz von etwa 1% aufweisen, sodass die vom Programm berechnete Optimierung der Nachkommastellen von rein theoretischer Natur ist.

2 Umsetzung

Für den Umgang mit den Bauplänen wird ein komplexer Datentyp, eine Art Baumstruktur, konstruiert, der als Zeigerstruktur implementiert wird. Dieser ist beidseitig verschaltet, hat also zwei Zeiger auf die Widerstände, die ihn ersetzen, und einen Zeiger auf seinen Vorgänger. Darüber hinaus speichert er die Art der Verschaltung der zwei ihn ersetzenden Widerstände. Dafür müssen drei Zustände bereitgehalten werden: `unused`, wenn der Widerstand ein Endwiderstand ist und dadurch keine Nachfolger verschaltet, `seriell` und `parallel`.

Aus der Methode `main()` wird die Funktion `netzaufbau()` aufgerufen. Sie ersetzt den Widerstand `preresistor` durch zwei Nachfolger, und reduziert anschließend die Anzahl der noch zu ersetzenden Widerstände (`remainingnum`) um eins. In einer Schleife, die von null bis eins zählt, wird der Zustand des zu ersetzenden Widerstandes dabei immer einmal auf `seriell` und einmal auf `parallel` gesetzt. Anschließend werden zwei neue Widerstände instanziiert. Mit einer Zählschleife, die von null bis unter die Anzahl der bisher verbleibenden Widerstände zählt, werden die Anzahlen an Widerständen bestimmt, die von der mit den Nachfolgern rekursiv aufzurufenden gleichen Funktion noch ersetzt werden sollen. Dafür gilt, dass die Summe der beiden zu verteilenden Anzahlen immer gleich der noch verbleibenden Zahl ist. Sollte diese Zahl gleich null sein, findet kein erneuter rekursiver Aufruf statt. Es ist dann möglich, dass bereits alle zu verteilenden Endwiderstände verteilt wurden. Aus der Funktion allein lässt sich das jedoch nicht mit Gewissheit sagen, da die `netaufbau()`-Funktion baumartig aufgebaut ist. Um das festzustellen, wird die Funktion `go_back()` aufgerufen, die über die Rückwärtsverschaltung der Zeigerstruktur zurück in die Wurzel geht und ab dort die Methode `count()` zwecks Durchzählens aufruft. Wenn die angeforderte Zahl an Endwiderständen noch nicht verteilt wurde, wird zurückgesprungen und der Bauplan weiter aufgebaut. Andernfalls kombiniert eine ebenfalls rekursive Funktion in einem Array der Länge der Anzahl an Endwiderständen sämtliche Kombinationsmöglichkeiten der Werte aus der sogenannten Grabbelkiste. Das geschieht, indem mithilfe einer Zählschleife durch die sogenannte Grabbelkiste durchgezählt wird. In einem Array mit der Länge der Anzahl der Werte in der sogenannten Grabbelkiste werden boolsche Werte gespeichert. Hier steht `false`, wenn dieser Wert noch nicht verwendet wurde, und `true`, wenn er bereits in Verwendung ist. Weil keine Widerstände doppelt verwendet werden dürfen, wird letzterer Fall für das weitere Kombinieren außer Acht gelassen. Andernfalls wird der Widerstand reserviert, indem sein Wert im Array mit den boolschen Werten auf `true` gesetzt wird. In einen Array der Länge der Anzahl der zu verwendenden Widerstände wird der Wert an die Stelle der Tiefe in der Rekursion gesetzt, und die Aufgabe rekursiv an die nächste Tiefe weitergegeben. Immer wenn auf diese Art so viele Werte kombiniert wurden, wie Endwiderstände versorgt werden müssen, wird die Funktion `weiterreichen()` aufgerufen, die den

Array rekursiv an die nachfolgenden Widerstände so aufteilt und weitergibt, dass für die Endwiderstände noch genau ein Element enthalten ist: ihr Widerstandswert. Die `calculate()`-Methode geht rekursiv den Baum hoch und berechnet beim zurückgehen die Widerstandswerte der Ersatzwiderstände regelkonform. So befinden wir uns wieder in der Rekombinationsfunktion, die jetzt den Betrag der Differenz zwischen Widerstand des Ersatzbauteils und dem angestrebten Wert bildet und mit dem bisher minimalen Wert vergleicht. Wenn diese Differenz geringer ist als der bisher minimale Wert, wird dieser durch jene ersetzt und der Zeiger `bestcombi`, der stets den besten Bauplan speichert, erhält, nachdem der vorherige Inhalt suber gelöscht wurde, den gerade berechneten Bauplan.

Auf die Art werden alle möglichen Baupläne mit ihren Wertkombinationsmöglichkeiten ausprobiert, sofern die Differenz noch nicht auf null minimiert wurde. Die Funktion `main()` führt diese Kombination mit zwei, drei und vier Widerständen herbei, wobei sie immer dann vorzeitig abbricht, wenn die Differenz auf null gefallen ist. Wenn die Hinzunahme eines weiteren Widerstandes gegenüber dem vorigen besten Ergebnis keinen Vorteil bringt, wird hierzu auch keine Lösung ausgegeben, da die zusätzliche Verwendung eines Widerstandes nicht sinnvoll wäre.

Zur Formatierung des Ausdruckes des Bauplans kann noch gesagt werden, dass die Klammern immer die Verschaltung dieser beiden Widerstände hervorhebt, wobei zwischen ihnen entweder `=` für eine Parallelschaltung steht, und `-` für eine serielle. Bei dem eingangs angeführten Beispiel mit drei Endwiderständen gibt der Computer also Folgendes aus: `((330-270)=1000)`.

Es sollte an dieser Stelle noch darauf hingewiesen werden, dass für die Funktionstüchtigkeit des Programmes das Vorhandensein einer Datei »widerstaende.txt« im gleichen Verzeichnis unabdingbar ist, die die Werte der in der sogenannten Grabbelkiste vorhandenen Widerstände enthält.

3 Beispiele

Im Folgenden werden die Programmausgaben für die Beispiele der BWINF-Seite abgedruckt:

```
Welchen Widerstandswert soll ich durch Kombination erreichen?
500
Die beste Kombination mit 2 Widerstaenden ist folgende:
(4700=560)
'-> 500.380228. Das bedeutet eine Differenz von nur 0.380228

Die beste Kombination mit 3 Widerstaenden ist folgende:
((180-100)-220)
'-> 500.000000. Das bedeutet eine Differenz von nur 0.000000

Ich muss nicht weiterrechnen.
Das Programm ist beendet.


Welchen Widerstandswert soll ich durch Kombination erreichen?
140
Die beste Kombination mit 2 Widerstaenden ist folgende:
(220=390)
'-> 140.655738. Das bedeutet eine Differenz von nur 0.655738

Die beste Kombination mit 3 Widerstaenden ist folgende:
((180=820)=2700)
'-> 139.949431. Das bedeutet eine Differenz von nur 0.050569

Die beste Kombination mit 4 Widerstaenden ist folgende:
(((180-1800)-120)=150)
'-> 140.000000. Das bedeutet eine Differenz von nur 0.000000

Das Programm ist beendet.


Welchen Widerstandswert soll ich durch Kombination erreichen?
314
Die beste Kombination mit 2 Widerstaenden ist folgende:
(6800=330)
'-> 314.726508. Das bedeutet eine Differenz von nur 0.726508

Die beste Kombination mit 3 Widerstaenden ist folgende:
((1800-4700)=330)
'-> 314.055637. Das bedeutet eine Differenz von nur 0.055637

Die beste Kombination mit 4 Widerstaenden ist folgende:
(((470=1200)-120)=1000)
'-> 313.999343. Das bedeutet eine Differenz von nur 0.000657

Das Programm ist beendet.


Welchen Widerstandswert soll ich durch Kombination erreichen?
315
Die beste Kombination mit 2 Widerstaenden ist folgende:
(6800=330)
'-> 314.726508. Das bedeutet eine Differenz von nur 0.273492

Die beste Kombination mit 3 Widerstaenden ist folgende:
```

```
((330-390)=560)
'-> 315.000000. Das bedeutet eine Differenz von nur 0.000000

Ich muss nicht weiterrechnen.
Das Programm ist beendet.


Welchen Widerstandswert soll ich durch Kombination erreichen?
1620
Die beste Kombination mit 2 Widerstaenden ist folgende:
(1500-120)
'-> 1620.000000. Das bedeutet eine Differenz von nur 0.000000

Ich muss nicht weiterrechnen.
Das Programm ist beendet.


Welchen Widerstandswert soll ich durch Kombination erreichen?
2719
Die beste Kombination mit 2 Widerstaenden ist folgende:
(1500-1200)
'-> 2700.000000. Das bedeutet eine Differenz von nur 19.000000

Die beste Kombination mit 3 Widerstaenden ist folgende:
((1000-220)-1500)
'-> 2720.000000. Das bedeutet eine Differenz von nur 1.000000

Die beste Kombination mit 4 Widerstaenden ist folgende:
(((180=220)-820)-1800)
'-> 2719.000000. Das bedeutet eine Differenz von nur 0.000000

Das Programm ist beendet.


Welchen Widerstandswert soll ich durch Kombination erreichen?
4242
Die beste Kombination mit 2 Widerstaenden ist folgende:
(330-3900)
'-> 4230.000000. Das bedeutet eine Differenz von nur 12.000000

Die beste Kombination mit 3 Widerstaenden ist folgende:
((2700=390)-3900)
'-> 4240.776699. Das bedeutet eine Differenz von nur 1.223301

Die beste Kombination mit 4 Widerstaenden ist folgende:
(((180=120)-3900)-270)
'-> 4242.000000. Das bedeutet eine Differenz von nur 0.000000

Das Programm ist beendet.
```

4 Quellcode

Die Funktion `main()`:

```
int main()
{
  auslesen(); // Die Textdatei wird gelesen.
  printf("Welchen Widerstandswert soll ich durch Kombination erreichen?\n"); //
    Nutzerabfrage
  scanf("%lf", &angestrebt);
  totalres = MAXres;
  for (int i = 0; i < MAXgrabbel; i++)
    verwendet[i] = false;
  for (int i = 2; i <= MAXres; i++) // Diese Schleife zaehlt von 2 bis zur maximalen Zahl
     von Widerstaenden, um diese jeweils auszuprobieren.
  {
    totalres = i;
    Resistor *startresistor = new_Resistor();
    netzaufbau(startresistor, i - 1); // Die Funktion fuer den Aufbau des Bauplans wird
    aufgerufen.
    if (count(bestcombi) != i) // Wenn die Zahl der verwendeten Widerstaende in der
    besten Kombination nicht erhoeht wurde, bringt eine Verwendung von mehr Widerstaenden
     keinen Vorteil.
      printf("Eine Kombination mit %d Widerstaenden bringt keinen Vorteil.\n", i);
    else
    {
      printf("Die beste Kombination mit %d Widerstaenden ist folgende:\n", i); //
    Ansonsten wird die Kombination ausgedruckt.
      printway2(bestcombi);
      double erg = bestcombi->resistance;
      double differenz = angestrebt - erg;
      if (differenz < 0) differenz = -differenz;
      printf("\n'-> %lf. Das bedeutet eine Differenz von nur %lf\n\n", erg, differenz);
      if (i != MAXres && differenz == 0) // Wenn bereits eine Abweichung von genau 0
    erreicht wurde, ist ein weiteres Rechnen mit mehr Widerstaenden nicht noetig.
      {
        printf("Ich muss nicht weiterrechnen.\n");
        break;
      }
    }
        remove_resistor(startresistor);
  }
  printf("Das Programm ist beendet.\n");
  return 0;
}
```

Die für den Aufbau der Baupläne notwendige Funktion `netzaufbau()`

```
void netzaufbau(Resistor *preresistor, int remainingnum) // Diese Funktion dient dazu,
    einen Bauplan fuer die Widerstaende zu erstellen, der noch keine Werte beinhaltet.
{
  remainingnum--; // Weil ein Widerstand durch zwei ersetzt werden soll, muss die Zahl
    der noch zu verbauenden Widerstaende verringert werden.
  for (int i = 0; i < 2; i++)
  {
    preresistor->con = (i == 0) ? seriell : parallel; // Die zwei Moeglichkeiten der
    Kombination werden durch eine Zaehlschleife, die von null bis eins zaehlt,
    ausprobiert.
    preresistor->r1 = new_Resistor(); // Die beiden Zeiger bekommen eine neue
    Zeigerstruktur
    preresistor->r2 = new_Resistor();
    preresistor->r1->pre = preresistor; // Die Nachfolger bekommen ihren Vorgaenger
    eingespeichert.
    preresistor->r2->pre = preresistor;
    if (remainingnum > 0) // Falls noch weitere Widerstaende zu verbauen sind, teilt eine
     Zaehlschleife die verbleibenden auf, sodass die Aufgabe rekursiv weitergegeben
    werden kann.
    {
      for (int j = 0; j < remainingnum; j++)
      {
        if (remainingnum - j > 0 && bestlow != 0)
          netzaufbau(preresistor->r1, remainingnum - j);
```

```
        if (j > 0 && bestlow != 0)
          netzaufbau(preresistor->r2, j);
      }
    }
    else
      go_back(preresistor);

      remove_resistor(preresistor->r1); // Die Nachfolger werden wieder sauber geloescht.
      remove_resistor(preresistor->r2);
    preresistor->r1 = NULL;
    preresistor->r2 = NULL;
  }
}
```

Die Funktion `go_back()` geht wieder in den Anfang des Datentyps:

```
void go_back(Resistor *myresistor) // Diese Funktion dient dazu, von einem gebauten Widerstandsbauplan an seinen Anfang zurueckzugehen.
{
  if (myresistor->pre != NULL)
    go_back(myresistor->pre); // Wenn der Anfang noch nicht erreicht ist, wird die Aufgabe rekursiv weitergegeben.
  else
  {
    if (count(myresistor) == totalres) // Wenn die Anzahl der verbauten Transistoren der angestrebten Anzahl entspricht, beginnt die Kombination der Werte.
      recombine(myresistor, 0);
  }
}
```

Das Kombinieren der Werte und die Überprüfung, ob ein neues Minimum vorliegt, übernimmt diese Funktion:

```
void recombine(Resistor *startresistor, int depth) // Diese Funktion wird dann aufgerufen, wenn ein fertiger Bauplan erstellt wurde. Sie kombiniert durch rekursive Aufrufe die Widerstandswerte.
{
  if (depth < totalres)
  {
    for (int i = 0; i < totalgrabbel; i++) // Wenn noch nicht fuer jeden Endwiderstand ein Wert zugeordnet wurde, zaehlt eine Schleife durch die Grabbelkiste
    {
      if (verwendet[i] == false) // Wenn der entsprechende Widerstand noch nicht verwendet wurde, wird er ausprobiert
      {
        verwendet[i] = true; // Er wird reserviert, damit er nicht doppelt verwendet wird
        reihenfolge[depth] = grabbelkiste[i]; // Der Endwiderstand der entsprechenden Tiefe bekommt den Wert
        if (bestlow != 0) recombine(startresistor, depth + 1); // Die Aufgabe wird rekursiv weitergegeben
        verwendet[i] = false; // Die Reservierung des Widerstandes wird aufgeloest
      }
    }
  }
  else
  {
    weiterreichen(startresistor, reihenfolge); // Wenn so viele Werte wie Widerstaende kombiniert wurden, werden die Werte an die Endwiderstaende weitergegeben.
    double erg = calculate(startresistor); // Der Widerstand des Bauplans mit den Werten wird berechnet.
    double differenz = angestrebt - erg;
    if (differenz < 0) differenz = -differenz; // Der Betrag der Differenz wird gebildet.
    if (differenz < bestlow)
    {
      remove_resistor(bestcombi); // Wenn die Differenz geringer ist als die bisher geringste Differenz, wird der Speicher, der den Bauplan speichert, freigegeben.
      bestcombi = clone_Resistor(startresistor, NULL); // Anschliessend wird er mit dem neuen Bauplan besetzt.
      bestlow = differenz; // Die geringste Differenz wird angepasst.
    }
```

```
  }
}
```

Regelkonform addiert diese Funktion die Werte für die Ersatzwiderstände auf:

```
double calculate(Resistor *myresistor) // Diese Funktion berechnet rekursiv die Werte der Ersatzwiderstaende
{
  if (myresistor->r1 != NULL)
  {
    double wider1 = calculate(myresistor->r1);
    double wider2 = calculate(myresistor->r2);

    if (myresistor->con == seriell)
      myresistor->resistance = wider1 + wider2;
    else
      myresistor->resistance = wider1 * wider2 / (wider1 + wider2);
  }
  return myresistor->resistance;
}
```

Damit die Werte in die Endwiderstände gelangen, gibt diese Funktion sie dorthin:

```
void weiterreichen(Resistor *myresistor, int einzelwiderstaende[]) // Funktion, die einen Array der Widerstandswerte an die untergeordneten Widerstaende weitergibt
{
  if (myresistor->r1 != NULL)
  {
    int links[MAXres]; // Falls der aktuelle Widerstand kein Endwiderstand ist, werden neue Arrays zur Weitergabe deklariert.
    int rechts[MAXres];
    for (int i = 0; i < myresistor->r1->untergeordnete; i++) // Diese Arrays werden mit jeweils so vielen Widerstandswerten aufgefuellt, wie die Nachgeordneten untergeordnete Widerstaende besitzen
      links[i] = einzelwiderstaende[i];
    for (int i = 0; i < myresistor->r2->untergeordnete; i++)
      rechts[i] = einzelwiderstaende[i + myresistor->r1->untergeordnete];
    weiterreichen(myresistor->r1, links); // Die Arrays mit den Widerstandswerten werden an die Nachgeordneten weitergegeben
    weiterreichen(myresistor->r2, rechts);
  }
  else // Falls der aktuelle Widerstand ein Endwiderstand ist, hat der Array noch genau ein Element, das der Wert des Endwiderstandes ist.
    myresistor->resistance = (double)einzelwiderstaende[0];
}
```