

Physik GFS - Schaltkreise

Name:
Lars Noack

14. März 2022

Inhaltsverzeichnis

1	Physikalische Grundlagen	1
1.1	Ohmsches Gesetz	1
1.2	Kirchofsche Regeln	1
	Knotenregel	1
	Maschenregel	1
1.3	Widerstände in einer Reihenschaltung	1
	Spannung	1
	Widerstand	1
1.4	Widerstände in einer Parrallelschaltung	2
	Stromstärke	2
	Widerstand	2
2	Repräsentation eines Schaltkreises als Baum	2
2.1	Die Schaltung als ungerichteter Graph	2
2.2	Die Schaltung als Baum	2
3	die Eingabe des Schaltkreises	3
3.1	Speichern des Baums in einer Adjazenzliste	3
3.2	Eigentliche Eingabe	3
3.3	Details der Implementierung	3
3.4	Beispiel	4
4	Das Einlesen der Liste	4
4.1	Klassen	4
4.2	Die gewünschte Klassenstruktur erhalten	4
5	Des Rendern des Schaltplans	5
5.1	schemdraw	5
5.2	Implementierung	5
6	Berechnung der fehlenden Werte	5
6.1	Basis	6
6.2	Warum lässt der Algorithmus nichts aus?	6
7	Beispiel	7
7.1	Komplexer Schaltkreis	7
7.2	Aufgabe 1	8
7.3	Aufgabe 2	9
7.4	Aufgabe 3	10
7.5	Aufgabe 4	11
7.6	Aufgabe 16	12
7.7	Aufgabe 7	13
8	Quellcode	14
8.1	GUI	14

8.2	die Adjazenzliste verarbeiten	15
8.3	die Logik der Knoten	16

Ich gehe davon aus, dass die Aufgabe klar ist, sonst geht diese aus der Projektsite heraus.

1 Physikalische Grundlagen

Wir wollen die Stromstärke, Spannung und den Widerstand aller Widerstände und Teilwiderstände in einer Schaltung berechnen. Dafür brauchen wir Formeln.

1.1 Ohmsches Gesetz

Das Ohmsche Gesetz beschreibt den Zusammenhang von Stromstärke, Spannung und dem Widerstand in einem Widerstand bzw in einem Teilwiderstand.

Wenn $R = \text{Konstant}$ dann $U \sim I$

Die Formel dafür lautet $U = R \cdot I$ und kann umgeschrieben werden als $R = \frac{U}{I}$ und in $I = \frac{U}{R}$

1.2 Kirchhofsche Regeln

Die Kirchhofsche Regeln sind ein Set von Regeln, bestehend aus 2 Regeln, der Knotenregel und der Maschenregel. Diese beschreiben das Verhalten von Stromstärke bzw. Spannung in Reihen- bzw. Parallelschaltungen, unabhängig vom Widerstand.

Knotenregel Die Knotenregel besagt, dass die Spannung bei Knoten gleich ist. Ein Knoten bezeichnet in dem Fall eine Parallelschaltung. Vereinfacht sagt die Regel, dass alle Teilwiderstände in einer Parallelschaltung die gleiche Spannung haben.

$$U_{ges} = U_1 = U_2 = U_3 = U_4 = \dots$$

Maschenregel Die Maschenregel besagt, dass die Stromstärke bei allen Maschen gleich ist. Als eine Masche bezeichnet man hier eine Reihenschaltung. Vereinfacht sagt die Regel, dass alle Teilwiderstände in einer Reihenschaltung die gleichen Stromstärken haben.

$$I_{ges} = I_1 = I_2 = I_3 = I_4 = \dots$$

1.3 Widerstände in einer Reihenschaltung

Dies beschreibt das Verhalten der Stromstärke, der Spannung und des Widerstandes in einer Reihenschaltung.

Spannung $U_{ges} = U_1 + U_2 + U_3 + U_4 + \dots$

Widerstand $R_{ges} = R_1 + R_2 + R_3 + R_4 + \dots$

1.4 Widerstände in einer Parallelschaltung

Dies beschreibt das Verhalten von der Stromstärke, der Spannung und des Widerstandes in einer Parallelschaltung.

Stromstärke $I_{ges} = I_1 + I_2 + I_3 + I_4 + \dots$

Widerstand $\frac{1}{R_{ges}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4} + \dots$

2 Repräsentation eines Schaltkreises als Baum

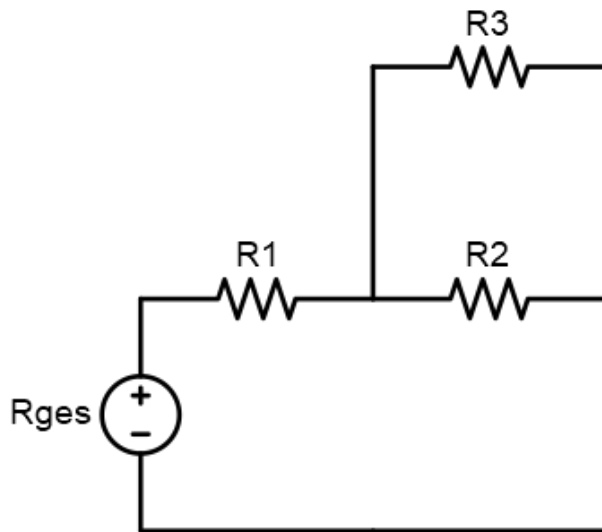
2.1 Die Schaltung als ungerichteter Graph

Eine Widerstand-Schaltung ist ein Graph. Also ein Netz aus Knoten, die mit Kanten verbunden sind. Die Knoten repräsentieren die Widerstände, wobei die Kanten die Verbindungen zwischen den Widerständen repräsentieren. Eigentlich ist der Graph ein gerichteter Graph, da der Strom nur in eine Richtung fließt. Jedoch ist die Fließrichtung für dieses Projekt irrelevant, wodurch wir auch einen ungerichteten Graphen nehmen können.¹

2.2 Die Schaltung als Baum

Ein Baum ist eine Datenstruktur in der Informatik. Der Unterschied zu einem Graph ist, dass ein Baum hierarchisch angeordnet ist und von oben nach unten geht. Also können auch bei einem ungerichteten Baum untere Knoten sich nicht mehr mit den Oberen verbinden

[https://de.wikipedia.org/wiki/Baum_\(Datenstruktur\)](https://de.wikipedia.org/wiki/Baum_(Datenstruktur))



Jetzt ist eine Schaltung so aber noch kein Baum, da sich ein Schaltkreis immer in einem Kreis schließt, was bei einem Baum verboten ist. Um aber zu erklären, wie ein Baum jede Widerstand-Schaltung repräsentieren kann, ziehe ich ein Beispiel hinzu (oben)².

Die Wurzel des Baumes³ ist eine Reihenschaltung. Dieser hat die Kinder R_1 und R_2, R_3 welche die Teilwiderstände in der Reihenschaltung sind.

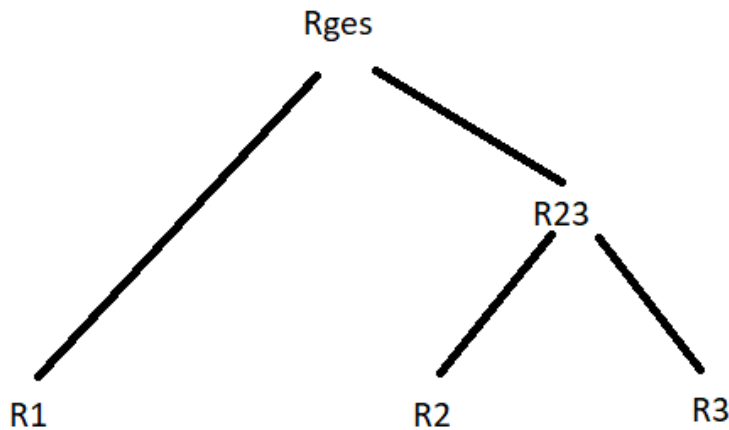
R_2, R_3 , das Kind der Wurzel, ist eine Parallelschaltung mit den Kindern R_2 und R_3 .

Dies kann man bis ins unendlich Komplexe skalieren.

¹Ein Graph ist gerichtet, wenn die Kanten nur in eine Richtung gehen und ungerichtet, wenn sie in beide Richtungen gehen. https://de.wikipedia.org/wiki/Gerichteter_Graph

²Die Grafik hat mein Programm generiert

³Der oberste Knoten



So würde der Baum dann aussehen.

3 die Eingabe des Schaltkreises

3.1 Speichern des Baums in einer Adjazenzliste

Graphen und Bäume werden häufig in einer Adjazenzliste gespeichert. Das heißt, man schreibt die Knoten, die von einer Kante verbunden sind, nebeneinander. Um besser zu beschreiben, was eine Adjazenzliste ist, ziehe ich das obere Beispiel des Baumes hinzu und erstelle die Adjazenzliste davon.⁴

Reihenschaltung R123, Widerstand R1
Reihenschaltung R123, Parallelschaltung R23
Parallelschaltung R23, Widerstand R2
Parallelschaltung R23, Widerstand R3

3.2 Eigentliche Eingabe

Jetzt reicht für die Eingabe keine einfache Adjazenzliste, da man nicht nur die Widerstände und Schaltungen angeben muss, sondern auch die angegebenen Werte der Widerstände und die der Teilwiderstände. Das heißt, ich brauche ein Separator zwischen den Knoten, und einen Separator, um die angegebenen Werte in den Knoten von den jeweiligen Verbindungsknoten zu unterscheiden.

Als Separator für die Knoten habe ich eine freie Zeile verwendet. Um in einem Knoten zwischen angegebenem Wert und Verbindung zu unterscheiden, rückt man die angegebenen Werte ein, und die Verbindungen nicht.

3.3 Details der Implementierung

- Wenn man einen Kommentar⁵ hinzufügen will macht man dass mit '//', inspiriert von Programmiersprachen wie C, C++, Java, JavaScript....
- Um den Typ eines Knoten anzugeben, schreibt man in der ersten Zeile nach dem Namen hinter dem Komma entweder R (resistor), P (parallel), S (series)
- Man muss nicht jeden Knoten angeben. Wenn der Knoten nur in einer Verbindung eines anderen Knoten erwähnt wird, wird angenommen, dieser Knoten ein Widerstand ohne angegebene Werte ist.
- Man muss nicht bei jedem Knoten alle Werte hinschreiben oder explizit erwähnen die fehlen, man muss lediglich z.B. R: 10 hinschreiben und dann weiß mein Programm, dass lediglich der Widerstand mit 10 Ohm angegeben wurde.

⁴Dies ist ein CSV comma separated format, das heißt die Knoten sind von einem Komma getrennt.

⁵Text, den mein Programm ignoriert um etwas zu beschreiben

- Diesen Text kann man in eine Datei schreiben und diese auslesen lassen, oder direkt in das Programm.

3.4 Beispiel

```
// Kommentar der Ignoriert wird.  
Rges, S  
    U: 24  
  
R1  
R23  
  
R23, P  
R2  
R3  
  
R2, R  
    R: 40  
  
R3, R  
    R: 60  
    I: 0.3
```

4 Das Einlesen der Liste

Vorab, der Code des Programmes der dies macht ist [hier](#) zu finden.

Aber bevor ich darauf eingehe wie ich die Liste einlese, muss ich erstmal kurz erklären was Klassen sind.

4.1 Klassen

Eine Klasse ist ein Container. In diesem Container können Variablen oder Listen gespeichert werden, Programmcode stehen, der diese Variablen oder Listen bearbeitet, und vieles mehr auf das ich nicht in Tiefe eingehen kann. Was ich also gemacht habe ist, dass jeder Knoten eine Instanz einer Klasse ist, in der eine Liste weiterer Knoten (den untergeordneten Knoten) ist.

Dies vereinfacht die Verarbeitung der Daten immens.

4.2 Die gewünschte Klassenstruktur erhalten

- Um jetzt aber diese Klassenstruktur zu erhalten, gehe ich erst einmal mit einer Schleife jeden Knoten der Datei durch.
- Bei jedem Knoten schaue ich, ob ich ihn schon unter dem Namen in der Liste abgespeichert habe.
- Wenn ich dies habe, ändere ich trotzdem die Art des Knotens zu der, die in der Datei angegeben ist. (Das wird Sinn machen)
- Wenn dies nicht so ist, dann speichere den Namen unter dem jeweiligen Typ.
- Dann speichere ich alle angegebenen Werte unter dem Namen des jeweiligen Knoten.
- Der letzte wichtige Schritt ist das Durchgehen aller angegebenen Verbindungen des jeweiligen Knotens.
- Dabei speichere jede Verbindung ab, schaue aber auch, ob der Name des verbundenen Knotens schon abgespeichert ist.
- Sollte dies nicht der Fall sein, dann speichere ich ihn unter dem Typ Widerstand ohne Werte ab.⁶

Dann kann man diese Klassen einfach erstellen.

Wenn dieser Programmstruktur nicht gefolgt werden konnte, ist dies nicht schlimm sondern verständlich. Aber um nochmal zusammenzufassen:

jetzt haben wir diesen Baum, und können mit ihm arbeiten

⁶Deshalb war es auch wichtig bei jedem angegebenen Knoten den Typ zu ändern, ist der Knoten schon abgespeichert.

5 Des Rendern des Schaltplans

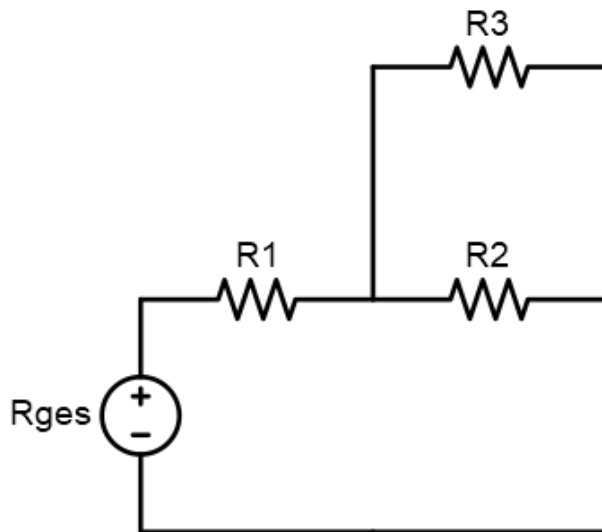
Dies war ein absolut notwendiger Schritt, da man sonst keine Möglichkeit hat zu überprüfen, ob der Schaltkreis richtig eingegeben wurde.

5.1 schemdraw

Ursprünglich wollte ich dieses Programm als Website machen, hab ich dann nicht gemacht, da ich für das Web bzw. JavaScript nur eine Library⁷ gefunden habe, bei der das Beispielpogramm 1000 Zeilen lang war. Für Python habe ich eine schönere Library gefunden, [schemdraw](#).

Mit ihr konnte ich ein Objekt erstellen, und dann zu dem Objekt verschiedene Elemente wie Stromquelle, Widerstände, Kondensatoren, etc. hinzufügen, bei dem ich aber nur Widerstände und eine Stromquelle verwendet habe.

Dann konnte ich das als svg exportieren und musste es dann in png umwandeln.⁸



5.2 Implementierung

Auch wenn diese sehr simpel zu erklären ist, war sie unglaublich schwer zu programmieren.

Als erstes habe ich ein Schemdraw Objekt erstellt. Dann habe ich dies der Wurzel gegeben. Wenn ein Knoten dann dieses Objekt bekommen hat, hat dieser, wenn er ein Widerstand ist, einfach ein Widerstand hinzugefügt. Ist dieser aber eine Reihen- oder Parallelschaltung, dann gibt er das Objekt an alle Kinder weiter, und zeichnet dann nur die Verbindungen zwischen den zurückgegebenen Änderungen.

Der Grund warum das sooooo schwer war, waren die vielen edge cases, zum Beispiel woher der Knoten weiß wie lang die Verbindungslinien sein sollen.

6 Berechnung der fehlenden Werte

Vorab, der Programmcode dazu ist [hier](#) zu finden.

Für so etwas existiert kein Algorithmus, der bekannt ist. Deshalb musste ich meinen eigenen machen. Mein Algorithmus weist jedoch Ähnlichkeiten zum [Bellman-Ford](#) Algorithmus⁹ auf. Bei Bellman-Ford werden immer benachbarte Knoten 'geupdated', wenn sich der Wert eines Knoten verändert. Ich mache das prinzipiell genauso.

⁷Code, den andere geschrieben haben.

⁸das hat soooo viele Stunden gebraucht.

⁹Ein Algorithmus zum Finden des kürzesten Weges, der häufig im Netzwerkrouting verwendet wird, also etwas ganz anderes macht.

6.1 Basis

Nur nochmal zur Erinnerung. Ein Knoten kann sowol ein Widerstand, als auch eine Parallelschaltung oder Reihenschaltung sein. Wenn er eine Schaltung ist, hat er mehr als 1 Kinder¹⁰, und wenn der Knoten ein Widerstand ist, hat er kein Kind. Als erstes betrachtet mein Programm die Wurzel. Hier wird dann zuerst¹¹ für alle 3 Werte¹² die Anzahl an Kindern herausgefunden, bei denen dieser Wert fehlt. Wenn bei einem der Werte nur 1 Wert fehlt (Eingenommen dem der Wurzel), so wird dieser je nach Schaltung berechnet. Wie das berechnet wird, wurde schon in Physikalische Grundlagen gesagt. Dann wird nach der Maschen- bzw. Knotenregel sowol bei sich selbst geschaut ob dort etwas gemacht werden kann, als auch bei allen Kindern. Anschließend wird auch geprüft, ob man mit dem Ohmschen Gesetz etwas berechnen kann, und ggf. wird der Wert dann berechnet.

Dann wird jedes Kind des Knotens aufgefordert das Gleiche zu machen (außer der Knoten ist ein Widerstand). **Das stellt sicher, dass das, solange alle Widerstände mit der Wurzel verbunden sind, mit jedem durchgeführt wird.**

6.2 Warum lässt der Algorithmus nichts aus?

Der Grund warum der Algorithmus funktioniert, ist dass jeder Knoten bei jeder Änderung eines Wertes alle obrigen Schritte macht, und somit alle Werte, die mit der Änderung dieses einen Werts noch berechnen werden können, auch berechnet werden. Hier ist auch die Ähnlichkeit zum Bellman-Ford Algorithmus - sie ist aber ziemlich gering.

Natürlich prüfe ich nicht immer alle Regeln, das wäre vergleichsweise ineffizient, sondern prüfe die nur dann wenn es Sinn macht. Das aber genau zu beschreiben würde hier einfach den Rahmen sprengen.

¹⁰1 Kind wäre möglich aber sinnlos

¹¹Wenn der Knoten eine Schaltung ist

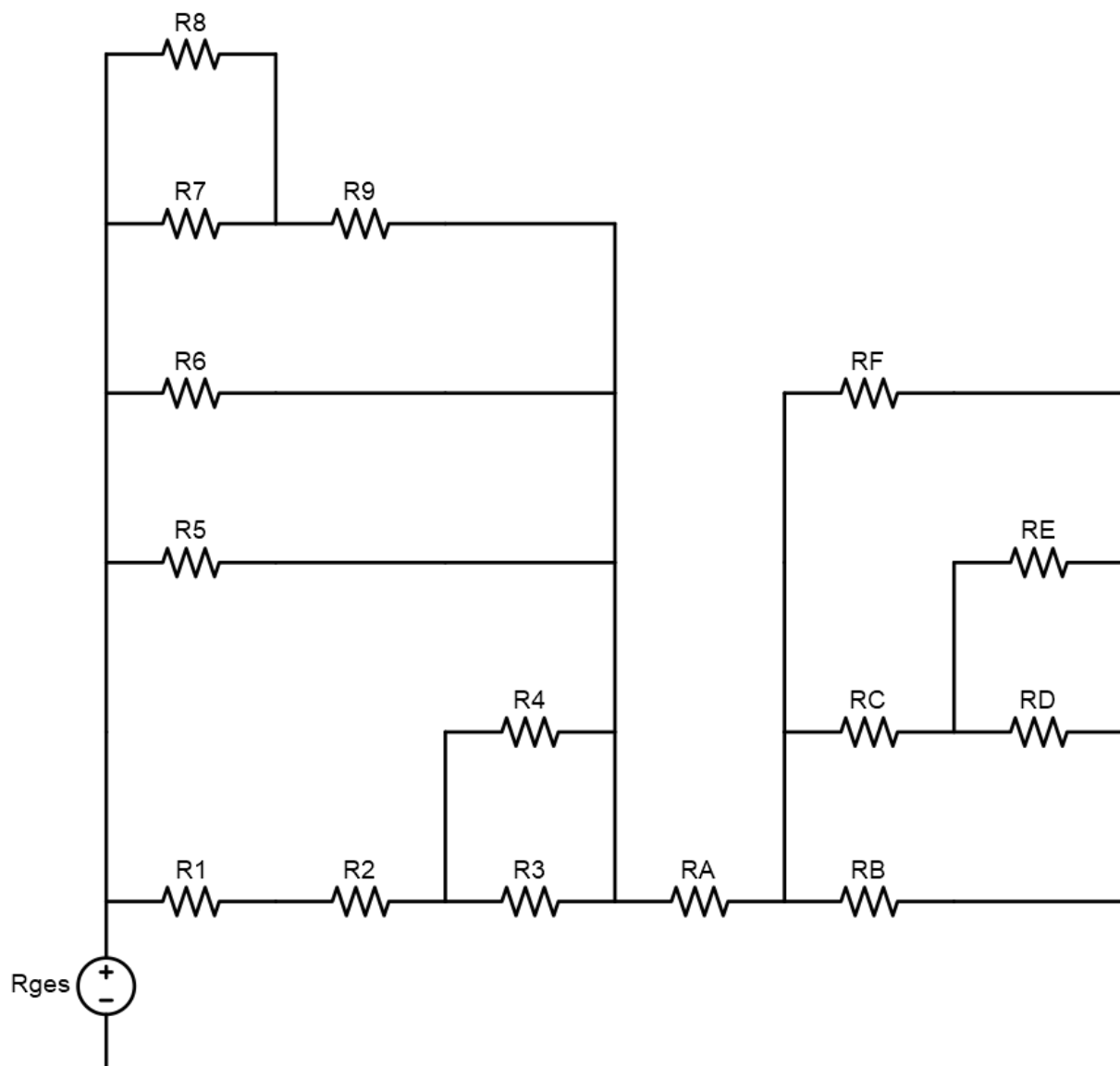
¹²U, R, I

7 Beispiel

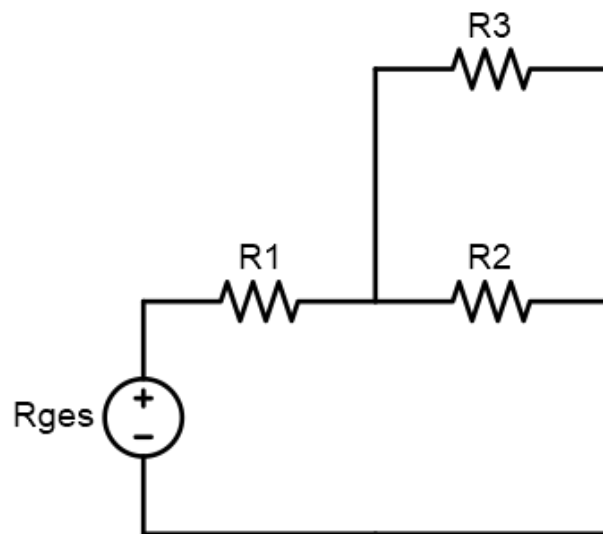
Bis auf das Beispiel um die Komplexität zu testen, habe ich die Beispiele von https://ulrich-rapp.de/stoff/fahrzeug/daten/ET_Ub_0hm.pdf genutzt, da man sich keine Fehler in der Aufgabe erlauben **DARF**, da sonst Werte falsch sind.

7.1 Komplexer Schaltkreis

Dies ist ein Beispiel ohne Werte, um zu zeigen, wie komplex die Schaltkreise werden können. Sie können sogar noch komplexer werden und mein Programm kommt damit klar.



7.2 Aufgabe 1



// 1

R_{ges} , S

U: 24

$R1$

$R23$

$R23$, P

$R2$

$R3$

$R2$, R

R: 40

$R3$, R

R: 60

I: 0.3

R_{ges} , Reihenschaltung

R: 32.00hm

U: 24.0V

I: 0.75A

$R1$, Widerstand

R: 8.00hm

U: 6.0V

I: 0.75A

$R23$, Parallelschaltung

R: 24.00hm

U: 18.0V

I: 0.75A

$R2$, Widerstand

R: 40.00hm

U: 18.0V

I: 0.45A

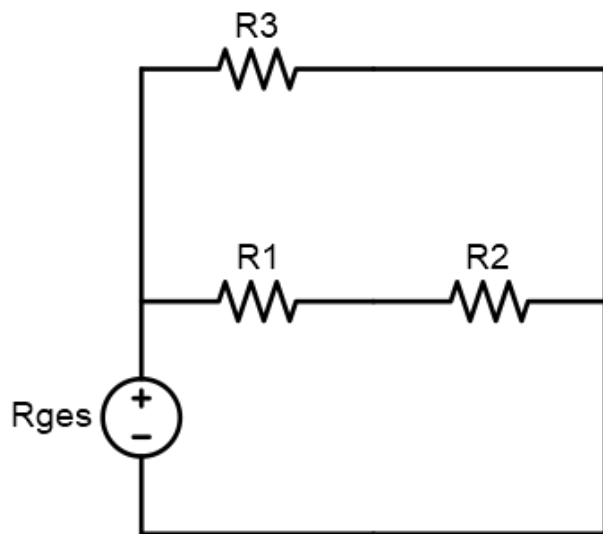
$R3$, Widerstand

R: 60.00hm

U: 18.0V

I: 0.3A

7.3 Aufgabe 2



```
// 2
Rges, P
  I: 0.7
R12
R3
```

```
R12, S
R1
R2
```

```
R1, R
  R: 20
```

```
R2, R
  R: 40
  U: 8
```

```
Rges, Parallelschaltung
  R: 17.143 Ohm
  U: 12.0V
  I: 0.7A
```

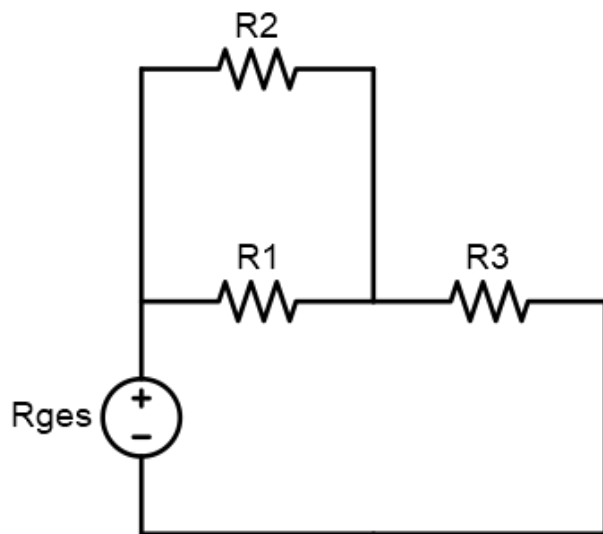
```
R12, Reihenschaltung
  R: 60.0 Ohm
  U: 12.0V
  I: 0.2A
```

```
R1, Widerstand
  R: 20.0 Ohm
  U: 4.0V
  I: 0.2A
```

```
R2, Widerstand
  R: 40.0 Ohm
  U: 8.0V
  I: 0.2A
```

```
R3, Widerstand
  R: 24.0 Ohm
  U: 12.0V
  I: 0.5A
```

7.4 Aufgabe 3



// 3

Rges, S
I: 8

R12
R3

R12, P
R1
R2

R1, R
I: 3

R2, R
R: 2

R3, R
R: 2

Rges, Reihenschaltung

R: 3.25 Ohm
U: 26.0V
I: 8.0A

R12, Parallelschaltung

R: 1.25 Ohm
U: 10.0V
I: 8.0A

R1, Widerstand

R: 3.333 Ohm
U: 10.0V
I: 3.0A

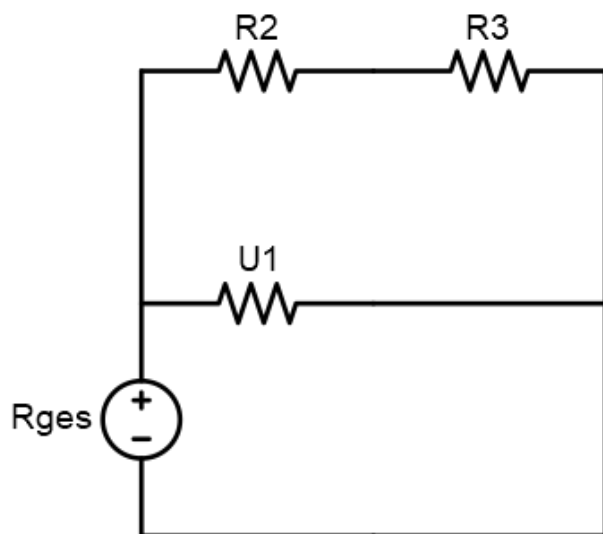
R2, Widerstand

R: 2.0 Ohm
U: 10.0V
I: 5.0A

R3, Widerstand

R: 2.0 Ohm
U: 16.0V
I: 8.0A

7.5 Aufgabe 4



// 4

R_{ges} , P
U: 42

$U1$
 $U23$

$U1$, R
R: 2.8

$U23$, S
 $R2$
 $R3$

$R2$, R
U: 12

$R3$, R
R: 1.25

R_{ges} , Parallelschaltung

R: 1.077 Ohm
U: 42.0V
I: 39.0A

$U1$, Widerstand

R: 2.8 Ohm
U: 42.0V
I: 15.0A

$U23$, Reihenschaltung

R: 1.75 Ohm
U: 42.0V
I: 24.0A

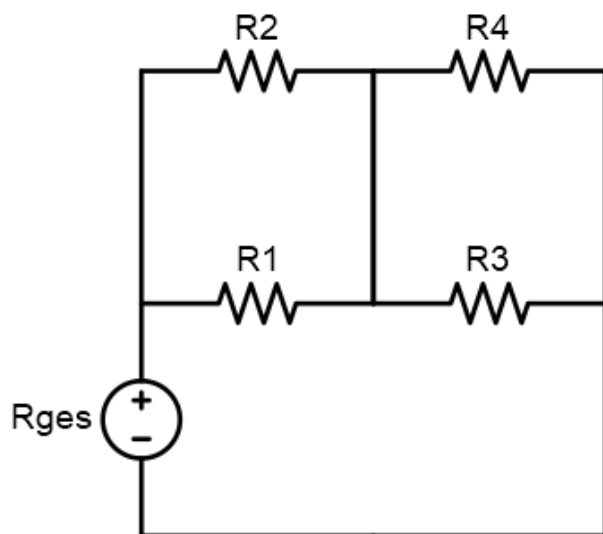
$R2$, Widerstand

R: 0.5 Ohm
U: 12.0V
I: 24.0A

$R3$, Widerstand

R: 1.25 Ohm
U: 30.0V
I: 24.0A

7.6 Aufgabe 16



// 16

R_{ges} , S

I: 20

R_{12}

R_{34}

R_{12} , P

R_1

R_2

R_1 , R

U: 10

R_2 , R

I: 5

R_{34} , P

R_3

R_4

R_{ges} , Reihenschaltung

R: 0.9 Ohm

U: 18.0V

I: 20.0A

R_{12} , Parallelschaltung

R: 0.5 Ohm

U: 10.0V

I: 20.0A

R_1 , Widerstand

R: 0.667 Ohm

U: 10.0V

I: 15.0A

R_2 , Widerstand

R: 2.0 Ohm

U: 10.0V

I: 5.0A

R_{34} , Parallelschaltung

R: 0.4 Ohm

U: 8.0V

I: 20.0A

R_3 , Widerstand

R: 0.667 Ohm

U: 8.0V

I: 12.0A

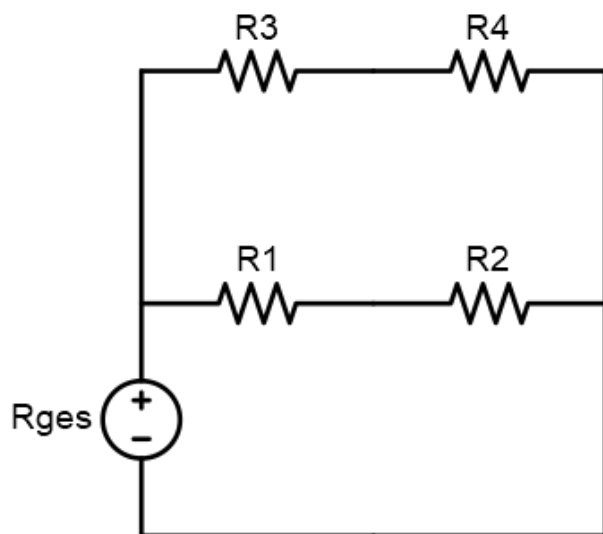
R_4 , Widerstand

R: 1.0 Ohm

U: 8.0V

I: 8.0A

7.7 Aufgabe 7



// 7

Rges, P
U: 42

R12
R34

R12, S
R1
R2

R34, S
R3
R4

R1, R
R: 1

R2, R
R: 2

Rges, Parallelschaltung

R: 2.1 Ohm
U: 42.0V
I: 20.0A

R12, Reihenschaltung

R: 3.0 Ohm
U: 42.0V
I: 14.0A

R1, Widerstand

R: 1.0 Ohm
U: 14.0V
I: 14.0A

R2, Widerstand

R: 2.0 Ohm
U: 28.0V
I: 14.0A

R34, Reihenschaltung

R: 7.0 Ohm
U: 42.0V
I: 6.0A

R3, Widerstand

R: 3.0 Ohm
U: 18.0V
I: 6.0A

R4, Widerstand

R: 4.0 Ohm
U: 24.0V
I: 6.0A

8 Quellcode

Den gesamten Quellcode, auch den der Website, finden Sie auf [Github](#) unter Creative Commons Lizenziert.

8.1 GUI

```

1  import read_circuit
3
5  import tkinter as tk
6  from tkinter import filedialog, Canvas
7  import os
8  from PIL import Image, ImageTk
9
10 class InputFrame:
11     def __init__(self, root):
12         self.root = root
13         self.fg_color = '#fff'
14         self.bg_color = '#bbb'
15
16         self.root.config(bg=self.bg_color)
17
18         self.string = ""
19         self.file_name = os.getcwd()
20
21         # Eingabe
22         self.open_file_dialog_button = tk.Button(self.root, text=self.file_name, command=self.file_dialog)
23         self.open_file_dialog_button.grid(row=0, column=0, sticky="NSEW", padx=5, pady=5)
24
25         self.text = tk.Text(self.root, height=20, width=30, undo=True, relief=tk.FLAT)
26         self.text.grid(row=1, column=0, sticky="NS", padx=5)
27         self.root.rowconfigure(1, weight=2)
28
29         self.calc_button = tk.Button(self.root, text="calculate_and_render", command=self.calculate)
30         self.calc_button.grid(row=2, column=0, sticky="EWS", padx=5, pady=5)
31
32         # Zeigt den Schaltplan am Ende
33         self.schematic = tk.Label(self.root, bg=self.bg_color)
34         self.schematic.grid(row=0, column=1, rowspan=3, sticky="NSEW", padx=5, pady=5)
35         self.root.columnconfigure(1, weight=2)
36
37         # Ausgabe
38         self.output_label = tk.Label(self.root, text="Ausgabe", bg=self.fg_color)
39         self.output_label.grid(row=0, column=2, sticky="NEW", padx=5, pady=5)
40
41         self.output_text = tk.Text(self.root, height=20, width=30, state="disabled", relief=tk.FLAT)
42         self.output_text.grid(row=1, column=2, rowspan=2, sticky="NS", padx=5, pady=(0, 5))
43
44         self.set_text()
45
46     def set_text(self):
47         if len(self.file_name) > 45:
48             display_file_name = "... " + self.file_name[-42:]
49             self.open_file_dialog_button.config(text=display_file_name)
50
51     def file_dialog(self):
52         file = tk.filedialog.askopenfile(mode="r", initialdir=os.getcwd())
53         if file is None:
54             return
55         self.file_name = file.name
56         self.set_text()
57         self.string = file.read()
58         self.text.delete('1.0', tk.END)
59         self.text.insert(tk.END, self.string)
60
61     def calculate(self):
62         print(self.root.winfo_width(), self.root.winfo_height())
63
64         save_to_str = self.file_name.split("/")[-1].split(".")[0]
65
66         circuit = read_circuit.Circuit(string=self.string, save_to=save_to_str)

```

```

67     img = Image.open(f'graphics/png/{save_to_str}.png')
69     im_width, im_height = img.size
70     elm_width, elm_height = self.schematic.wininfo_width(), self.schematic.wininfo_height()
71     if im_width/elm_width > im_height/elm_height:
72         ratio = im_height/im_width
73         im_height = elm_width * ratio
74         im_width = elm_width
75     elif im_width/elm_width < im_height/elm_height:
76         ratio = im_width/im_height
77         im_width = elm_height * ratio
78         im_height = elm_height
79     img = img.resize((int(im_width), int(im_height)))
81     self.tk_img = ImageTk.PhotoImage(img)
83     self.schematic.config(image=self.tk_img)
84     self.output_text.configure(state='normal')
85     self.output_text.delete('1.0', tk.END)
86     self.output_text.insert(tk.END, circuit.output)
87     self.output_text.configure(state='disabled')
89
90     root = tk.Tk()
91     root.title("Schaltplan_CLars_Noack")
92     # root.state("zoomed")
93     root.geometry("800x400")
95     input_frame = tk.Frame(root)
96     input_frame.grid(row=0, column=0)
97     InputFrame(root)
99     root.mainloop()

```

Listing 1: Sprache: Python

8.2 die Adjazenzliste verarbeiten

```

1  import re
2  import circuits
3
4
5  class Circuit:
6      def __init__(self, path="", string="", save_to=""):
7          file_str = ""
8
9          if path != "":
10             with open(path, "r") as circuit_file:
11                 string = circuit_file.read()
12
13             lines = string.split("\n")
14             for line in lines:
15                 x = re.findall("^\n", line)
16                 if not x:
17                     file_str += line + "\n"
18
19             connection_list = file_str.split("\n\n")
20             for i, connection in enumerate(connection_list):
21                 temp_connections = connection.split("\n")
22                 connections = []
23                 for connection_ in temp_connections:
24                     if connection_ != "":
25                         connections.append(connection_)
26                 connection_list[i] = connections
27
28             elements_dict = {}
29
30             root_name = ""
31
32             for connection in connection_list:
33                 connection[0] = connection[0].replace("_", "")

```



```

35         name, type_ = connection[0].split(",")

37         if root_name == "":
38             root_name = name

39         values_dict = {"U": -1, "R": -1, "I": -1}
40         values = []
41         child_names = []
42         for attribute in connection[1:]:
43             if re.findall("^_+", attribute) or re.findall("^\\t", attribute):
44                 attribute = attribute.replace("_", "").replace("\\t", "")
45                 values.append(attribute)
46             else:
47                 child_names.append(attribute)

49         for value in values:
50             key = value[0].upper()
51             values_dict[key] = float(value[2:])

53         elements_dict[name] = [child_names,
54                                circuits.Element(name, type_, voltage=values_dict["U"], resistance=
55                                                    current=values_dict["I"])]

57         for elem_key in list(elements_dict):
58             for child_name in elements_dict[elem_key][0]:
59                 if child_name in elements_dict:
60                     elements_dict[elem_key][1].add_child(elements_dict[child_name][1])
61                 else:
62                     elements_dict[elem_key][1].add_child(circuits.Element(child_name, "r"))

64         elements_dict[root_name][1].draw_as_root(save_to=save_to)
65         for i in range(10):
66             elements_dict[root_name][1].compute()

68         self.output = elements_dict[root_name][1].output_values()

71     if __name__ == '__main__':
72         Circuit("circuits/circuit2.cd")

```

Listing 2: Sprache: Python

8.3 die Logik der Knoten

```

2 import schemdraw
3 import schemdraw.elements as elm
4 schemdraw.use('svg')

6
7
8 class Element:
9     def __init__(self, name: str, type_str: str, resistance=-1.0, voltage=-1.0, current=-1.0):
10         type_str = type_str.lower()
11         types = {
12             "r": 0,
13             "s": 1,
14             "p": 2
15         }
16         self.type = types[type_str]
17         self.name = name

18         self.resistance = float(resistance)
19         self.voltage = float(voltage)
20         self.current = float(current)

22         self.child_Elements = []

24         self.on_change()

26
27     def ohmsches_gesetzt(self):
28         if self.resistance != -1 and self.current != -1 and self.voltage == -1:

```

Physik GFS - Schaltkreise

```
        self.voltage = self.resistance * self.current
30     return

32     if self.voltage != -1 and self.resistance != -1 and self.current == -1:
        if self.resistance != 0:
34             self.current = self.voltage / self.resistance
        else:
36             print(f"Der_Widerstand_bei_{self.name}_ist_0.")
        return

38     if self.voltage != -1 and self.current != -1 and self.resistance == -1:
        if self.current != 0:
40             self.resistance = self.voltage / self.current
        else:
42             print(f"Die_Stromspannung_bei_{self.name}_ist_0.")
44     return

46     # wenn alle Werte vorhanden sind, werden diese ueberprueft
    if self.voltage != -1 and self.current != -1 and self.resistance != -1:
48         if int(self.voltage) != int(self.resistance * self.current):
            print(f"Spannung[{self.resistance}]_!=_Widerstand[{self.resistance}]_*_Staerke[{self.cu

50
def maschen_knoten_children(self):
52     # wenn der Richtige wert bei diesem Objekt existiert, fuege ihn bei den Kindern hinzu.

54     if len(self.child_Elements) == 0:
        return

56
    if self.type == 1:
58         # Maschenregel
        if self.current == -1:
60             return

62         for child in self.child_Elements:
            child.set_current(self.current)
64         return

66     if self.type == 2:
        # Knotenregel
68         if self.voltage == -1:
            return

70
        for child in self.child_Elements:
72             child.set_voltage(self.voltage)
        return

74
def maschen_knoten_self(self):
76     # wenn der Richtige wert bei den Kindern existiert, fuege ihn bei sich selbst.

78     if len(self.child_Elements) == 0:
        return

80
    if self.type == 1:
82         # Maschenregel
        for child in self.child_Elements:
84             if child.current != -1:
                self.set_current(child.current)
86             return
        return

88
    if self.type == 2:
        # Knotenregel
90         for child in self.child_Elements:
92             if child.voltage != -1:
                self.set_voltage(child.voltage)
94         return

96
def set_voltage(self, voltage):
    if self.voltage != -1:
98         return -1
    self.voltage = voltage
100    self.on_change()
```

```

102     def set_current(self, current):
103         if self.current != -1:
104             return
105         self.current = current
106         self.on_change()

108     def set_resistance(self, resistance):
109         if self.resistance != -1:
110             return -1
111         self.resistance = resistance
112         self.on_change()

114     def on_change(self):
115         self.ohmsches_gesetzt()
116         self.maschen_knoten_children()

118     def compute(self):
119         if len(self.child_Elements) == 0:
120             return
121         if self.type == 0:
122             return

124         # get the amount of unknown values
125         unknown_voltage = 0
126         unknown_current = 0
127         unknown_resistance = 0
128         for child in self.child_Elements:
129             if child.voltage == -1:
130                 unknown_voltage += 1
131                 missing_voltage_elem = child
132             if child.current == -1:
133                 unknown_current += 1
134                 missing_current_elem = child
135             if child.resistance == -1:
136                 unknown_resistance += 1
137                 missing_resistance_elem = child

138         if self.type == 1:
139             # Reihenschaltung
140             if unknown_resistance == 0:
141                 total_resistance = 0
142                 for child in self.child_Elements:
143                     total_resistance += child.resistance
144                 self.set_resistance(total_resistance)

146             elif unknown_resistance == 1 and self.resistance != -1:
147                 total_resistance = 0
148                 for child in self.child_Elements:
149                     if child.resistance != -1:
150                         total_resistance += child.resistance
151                 missing_resistance_elem.set_resistance(self.resistance - total_resistance)

154             if unknown_voltage == 0:
155                 total_voltage = 0
156                 for child in self.child_Elements:
157                     total_voltage += child.voltage
158                 self.set_voltage(total_voltage)

160             elif unknown_voltage == 1 and self.voltage != -1:
161                 total_voltage = 0
162                 for child in self.child_Elements:
163                     if child.voltage != -1:
164                         total_voltage += child.voltage
165                 missing_voltage_elem.set_voltage(self.voltage - total_voltage)

166         elif self.type == 2:
167             # Parallelschaltung
168             if unknown_current == 0:
169                 total_current = 0
170                 for child in self.child_Elements:
171                     total_current += child.current
172                 self.set_current(total_current)
173
174

```

```

176         elif unknown_current == 1 and self.current != -1:
177             total_current = 0
178             for child in self.child_Elements:
179                 if child.current != -1:
180                     total_current += child.current
181
182             missing_current_elem.set_current(self.current - total_current)
183
184     self.maschen_knoten_self()
185     self.maschen_knoten_children()
186
187     for child in self.child_Elements:
188         child.compute()
189
190     def output_values(self, return_str=""):
191         type_descriptions = ["Widerstand", "Reihenschaltung", "Parallelschaltung"]
192         return_str += f"{self.name},{type_descriptions[self.type]}\n"
193         print(f"{self.name},{type_descriptions[self.type]}")
194         if self.resistance != -1:
195             print(f"R:{round(self.resistance, 3)}Ohm")
196             return_str += f"R:{round(self.resistance, 3)}Ohm\n"
197         if self.voltage != -1:
198             print(f"U:{round(self.voltage, 3)}V")
199             return_str += f"U:{round(self.voltage, 3)}V\n"
200         if self.current != -1:
201             print(f"I:{round(self.current, 3)}A")
202             return_str += f"I:{round(self.current, 3)}A\n"
203         return_str += "\n"
204         print("")
205
206         if self.type == 1 or self.type == 2:
207             for child in self.child_Elements:
208                 return_str = child.output_values(return_str=return_str)
209
210         return return_str
211
212     def add_child(self, child):
213         self.child_Elements.append(child)
214         self.maschen_knoten_self()
215
216     def draw(self, d: schemdraw.Drawing, latest_elem):
217         if self.type == 0:
218             d += (latest_elem := elm.Resistor().right().label(self.name).at(latest_elem.end))
219
220             return 1, latest_elem, 1
221
222         if self.type == 1:
223             steps = 0
224             height = 1
225
226             for child in self.child_Elements:
227                 r_steps, latest_elem, height = child.draw(d, latest_elem)
228                 steps += r_steps
229
230             return steps, latest_elem, height
231
232         if self.type == 2:
233             steps, last, heights = self.child_Elements[0].draw(d, latest_elem)
234             parallel_steps = [steps]
235             parallel_heights = [heights]
236             parallel_last = [last]
237
238             for child in self.child_Elements[1:]:
239                 for n in range(parallel_heights[-1]):
240                     d += (latest_elem := elm.Line().up().at(latest_elem.end))
241                     steps, last, height = child.draw(d, latest_elem)
242                     parallel_steps.append(steps)
243                     parallel_last.append(last)
244                     parallel_heights.append(height)
245
246             # get max steps
247             max_steps = 0

```

```

248         for step in parallel_steps:
249             if step > max_steps:
250                 max_steps = step
251
252         # draw rest of the circuit
253         for i, latest_elem in enumerate(parallel_last):
254             for n in range(max_steps - parallel_steps[i]):
255                 d += (latest_elem := elm.Line().right().at(latest_elem.end))
256             if i > 0:
257                 for n in range(parallel_heights[i]):
258                     d += (latest_elem := elm.Line().down().at(latest_elem.end))
259             else:
260                 latest_elem_r = latest_elem
261
262         current_height = 0
263         for heights in parallel_heights:
264             current_height += heights
265
266         return max_steps, latest_elem_r, current_height
267
268     def draw_as_root(self, save_to=""):
269         d = schemdraw.Drawing()
270
271         d += (latest_elem := elm.SourceV().up().label(self.name))
272
273         steps, latest_elem, height = self.draw(d, latest_elem)
274
275         d += (latest_elem := elm.Line().down().at(latest_elem.end))
276         for i in range(steps):
277             d += (latest_elem := elm.Line().left().at(latest_elem.end))
278
279         d.draw()
280         if save_to == "":
281             d.save('schematic.svg')
282         else:
283             d.save(f"graphics/png/{save_to}.png", dpi=300)
284             d.save(f"graphics/{save_to}.svg")

```

Listing 3: Sprache: Python