Physik GFS - Schaltkreise

Name:
Lars Noack

14. März 2022

Inhaltsverzeichnis

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. Dafuer 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 $R = Konstant$ 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 Kirchofsche Regeln

Die Kirchofsche 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. Parrallelschaltungen, unabhängig vom Widerstand.

Knotenregel Die Knotenregel besagt, dass die Spannung bei Knoten gleich ist. Ein Knoten bezeichnet in dem Fall eine Parrallelschaltung. Vereinfacht sagt die Regel, dass alle Teilwiderstände in einer Parrallelschaltung die gleiche Spannung haben.
$U_{ges} = U_1 = U_2 = U_3 = U_4 = ...$

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.
$I_{ges} = I_1 = I_2 = I_3 = I_4 = ...$

1.3 Widerstände in einer Reihenschaltung

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

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

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

1.4 Widerstände in einer Parrallelschaltung

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

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

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

2 Repräsentation eines Schaltkreises als Baum

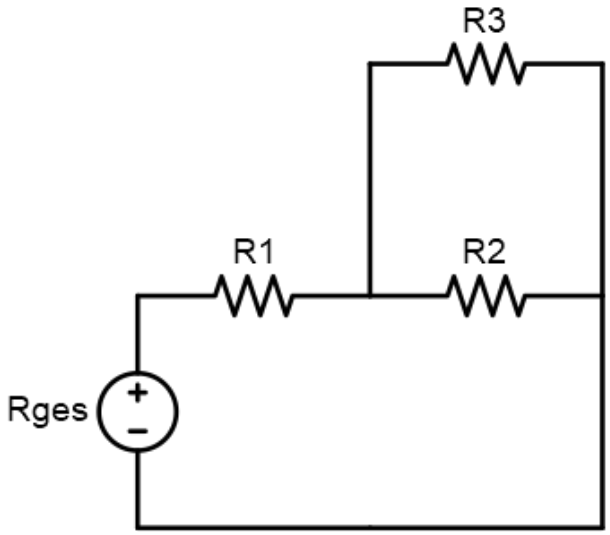
2.1 Die Schaltung als ungerichteter Graph

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 koennen.[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 koennen auch bei einem ungerichteten Baum untere Knoten sich nicht mehr mit den Oberen verbinden

`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 Baumes [3] ist eine Reihenschaltung. Dieser hat die Kinder R_1 und R_2, R_3 welche die Teilwiderstände in der Reihenschaltung sind.

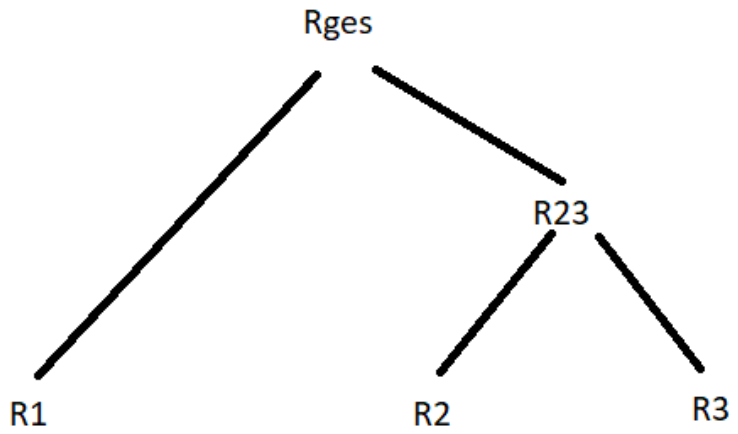
R_2, R_3, das Kind der Wurzel, ist eine Parrallelschaltung mit den Kindern R_2 und R_3.

Dies kann man bis ins unendlich Komplexe skalieren.

[1] Ein 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`

[2] Die Grafik hat mein Programm generiert

[3] Der 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, Parralelschaltung R23
Parralelschaltung R23, Widerstand R2
Parralelschaltung 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 Koten 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 Kommentar[5] 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.

[4] Dies ist ein CSV comma seperated format, das heißt die Knoten sind von einem Komma getrennt.
[5] Text, 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 gewünste 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

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

5 Des Rendern des Schaltplans

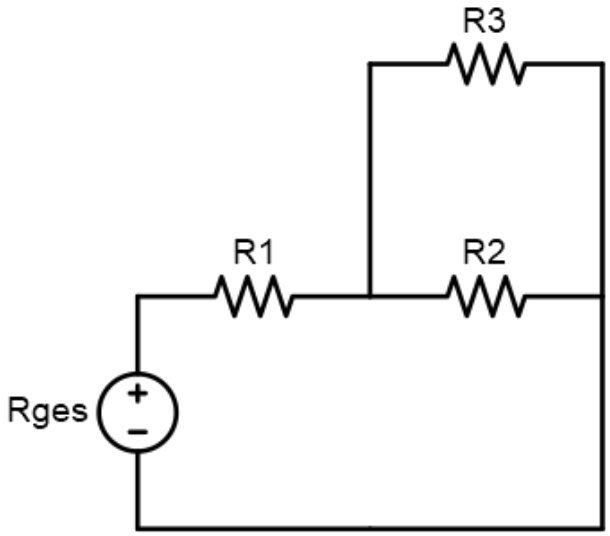
Dies war ein absolut notwendiger Schritt, da man sonst keine Möglichkeit hat zu überprüfen, ob der Schaltkreis richtig eingegeben wurde.

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 Library[7] gefunden habe, bei der das Beispielprogramm 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 soooooo 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 Algorithmus[9]auf. Bei Bellman-Ford werden immer benachbarte Knoten 'geupdated', wenn sich der Wert eines Knoten verändert. Ich mache das prinzipiell genauso.

[7] Code, den andere geschrieben haben.
[8] das hat soooo viele Stunden gebraucht.
[9] Ein 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 Kinder[10], und wenn der Knoten ein Widerstand ist, hat er kein Kind. Als erstes betrachtet mein Programm die Wurzel. Hier wird dann zuerst[11] für alle 3 Werte[12] 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 Ähnlickeit 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.

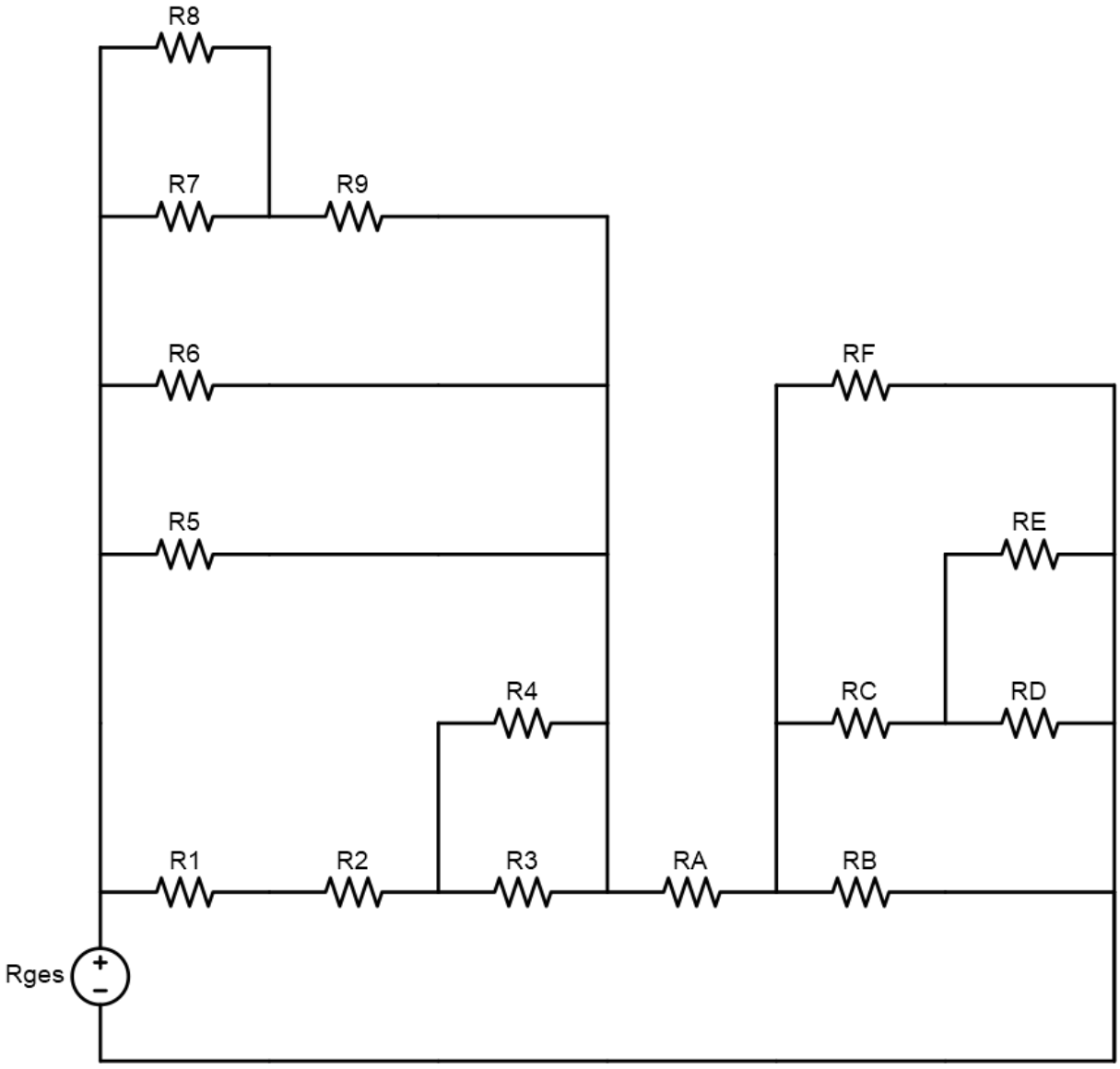
[10] 1 Kind wäre möglich aber sinnlos
[11] Wenn der Knoten eine Schaltung ist
[12] U, R, I

7 Beispiel

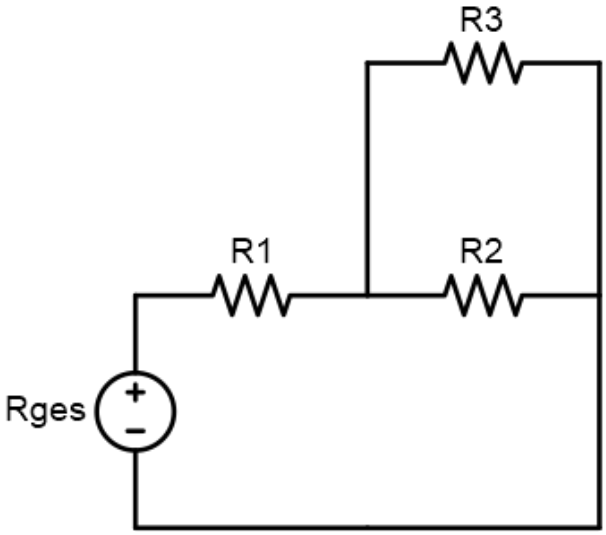
Bis auf das Beispiel um die Komplexität zu testen, habe ich die Beispiele von `https://ulrich-rapp.de/stoff/fahrzeug/daten/ET_Ub_Ohm.pdf` genutzt, da man sich keine Fehler in der Aufgabe erlauben **DARF**, da sonst Werte falsch sind.

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.



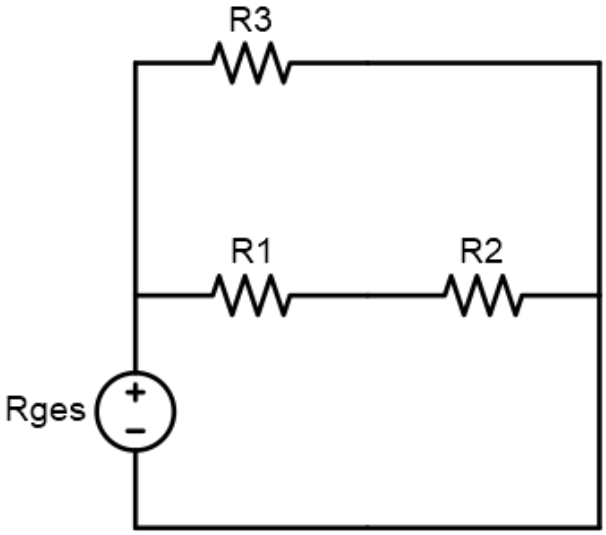
7.2 Aufgabe 1




```
// 1
Rges, S
    U: 24
R1
R23

R23, P
R2
R3

R2, R
    R: 40

R3, R
    R: 60
    I: 0.3
```

```
Rges, 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
```

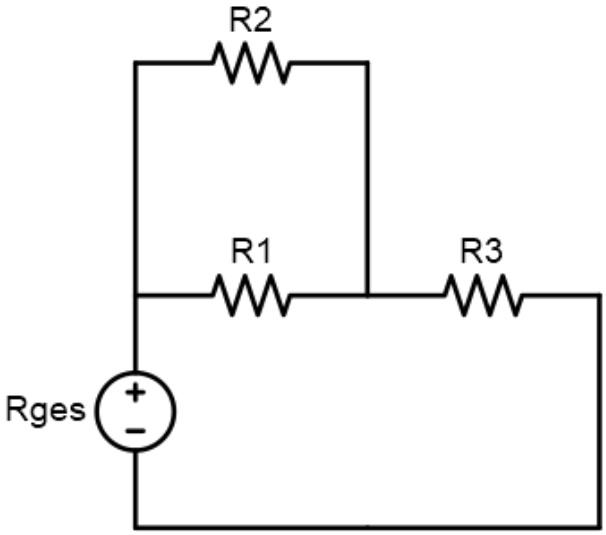
7.3 Aufgabe 2




```
// 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
```

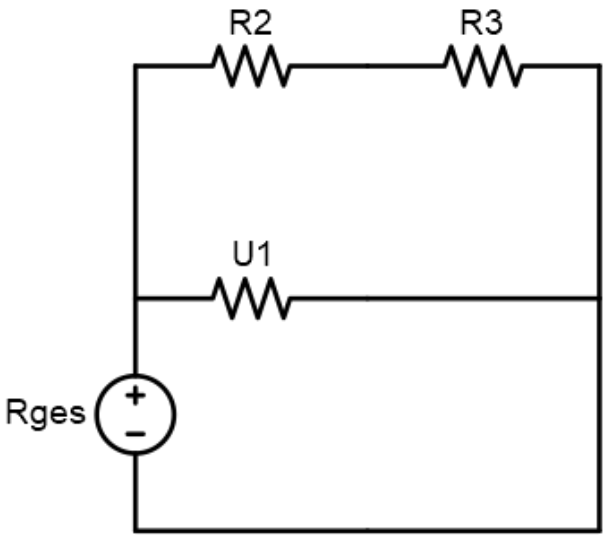
7.4 Aufgabe 3




```
// 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
```

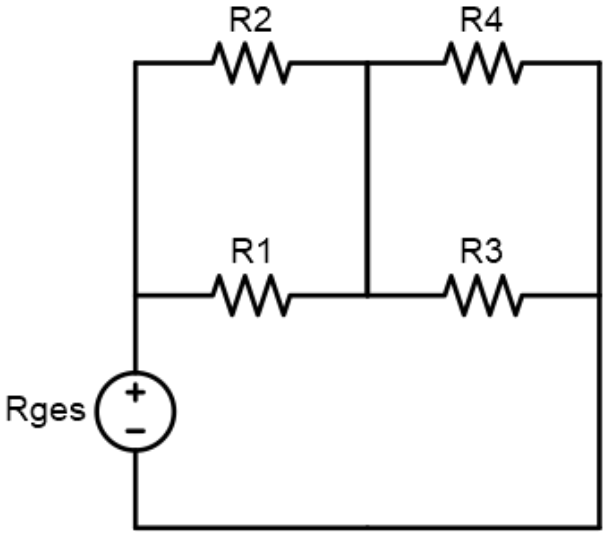
7.5 Aufgabe 4




```
// 4
Rges, P
    U: 42
U1
U23

U1, R
    R: 2.8

U23, S
R2
R3

R2, R
    U: 12

R3, R
    R: 1.25
```

```
Rges, 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
```

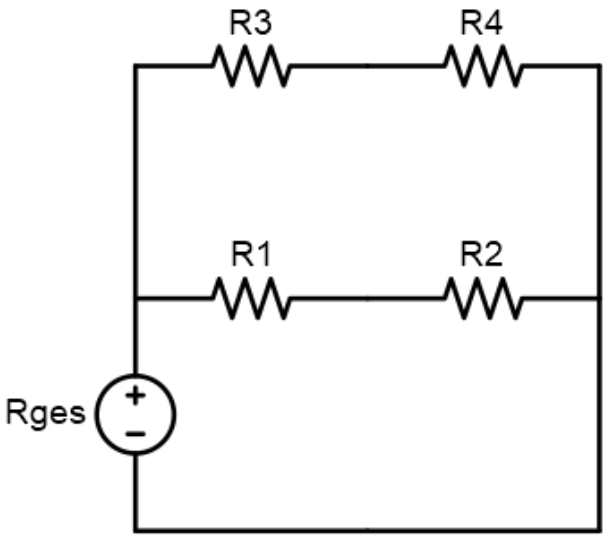
7.6 Aufgabe 16




```
// 16
Rges, S
    I: 20
R12
R34

R12, P
R1
R2

R1, R
    U: 10

R2, R
    I: 5

R34, P
R3
R4
```

```
Rges, Reihenschaltung
    R: 0.9 Ohm
    U: 18.0V
    I: 20.0A

R12, Parallelschaltung
    R: 0.5 Ohm
    U: 10.0V
    I: 20.0A

R1, Widerstand
    R: 0.667 Ohm
    U: 10.0V
    I: 15.0A

R2, Widerstand
    R: 2.0 Ohm
    U: 10.0V
    I: 5.0A

R34, Parallelschaltung
    R: 0.4 Ohm
    U: 8.0V
    I: 20.0A

R3, Widerstand
    R: 0.667 Ohm
    U: 8.0V
    I: 12.0A

R4, 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 Lizensiert.

8.1 GUI

```
1
import read_circuit
3
import tkinter as tk
5 from tkinter import filedialog, Canvas
import os
7 from PIL import Image, ImageTk

9 class InputFrame:
    def __init__(self, root):
11      self.root = root
        self.fg_color = '#fff'
13      self.bg_color = '#bbb'

15      self.root.config(bg=self.bg_color)

17      self.string = ""
        self.file_name = os.getcwd()
19
        # Eingabe
21      self.open_file_dialog_button = tk.Button(self.root, text=self.file_name, command=self.file_dialo
        self.open_file_dialog_button.grid(row=0, column=0, sticky="NSEW", padx=5, pady=5)
23
        self.text = tk.Text(self.root, height=20, width=30, undo=True, relief=tk.FLAT)
25      self.text.grid(row=1, column=0, sticky="NS", padx=5)
        self.root.rowconfigure(1, weight=2)
27
        self.calc_button = tk.Button(self.root, text="calculate and render", command=self.calculate)
29      self.calc_button.grid(row=2, column=0, sticky="EWS", padx=5, pady=5)

31      # Zeigt den Schaltplan am Ende
        self.schematic = tk.Label(self.root, bg=self.bg_color)
33      self.schematic.grid(row=0, column=1, rowspan=3, sticky="NSEW", padx=5, pady=5)
        self.root.columnconfigure(1, weight=2)
35
        # Ausgabe
37      self.output_label = tk.Label(self.root, text="Ausgabe", bg=self.fg_color)
        self.output_label.grid(row=0, column=2, sticky="NEW", padx=5, pady=5)
39
        self.output_text = tk.Text(self.root, height=20, width=30, state="disabled", relief=tk.FLAT)
41      self.output_text.grid(row=1, column=2, rowspan=2, sticky="NS", padx=5, pady=(0, 5))

43      self.set_text()

45  def set_text(self):
        if len(self.file_name) > 45:
47          display_file_name = "..." + self.file_name[-42:]
            self.open_file_dialog_button.config(text=display_file_name)
49
    def file_dialog(self):
51      file = tk.filedialog.askopenfile(mode="r", initialdir=os.getcwd())
        if file is None:
53          return
        self.file_name = file.name
55      self.set_text()
        self.string = file.read()
57      self.text.delete('1.0', tk.END)
        self.text.insert(tk.END, self.string)
59
    def calculate(self):
61      print(self.root.winfo_width(), self.root.winfo_height())

63      save_to_str = self.file_name.split("/")[-1].split(".")[0]

65      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
            elm_width, elm_height = self.schematic.winfo_width(), self.schematic.winfo_height()
71          if im_width/elm_width > im_height/elm_height:
                ratio = im_height/im_width
73              im_height = elm_width * ratio
                im_width = elm_width
75          elif im_width/elm_width < im_height/elm_height:
                ratio = im_width/im_height
77              im_width = elm_height * ratio
                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)
            self.output_text.configure(state='normal')
85          self.output_text.delete('1.0', tk.END)
            self.output_text.insert(tk.END, circuit.output)
87          self.output_text.configure(state='disabled')


89
root = tk.Tk()
91 root.title("Schaltplan CLars Noack")
  # root.state("zoomed")
93 root.geometry("800x400")

95 input_frame = tk.Frame(root)
  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
3 import circuits


5
class Circuit:
7     def __init__(self, path="", string="", save_to=""):
          file_str = ""
9
          if path != "":
11            with open(path, "r") as circuit_file:
                  string = circuit_file.read()
13
          lines = string.split("\n")
15        for line in lines:
              x = re.findall("^//", line)
17            if not x:
                  file_str += line + "\n"
19
          connection_list = file_str.split("\n\n")
21        for i, connection in enumerate(connection_list):
              temp_connections = connection.split("\n")
23            connections = []
              for connection_ in temp_connections:
25                if connection_ != "":
                      connections.append(connection_)
27            connection_list[i] = connections

29        elements_dict = {}

31        root_name = ""

33        for connection in connection_list:
              connection[0] = connection[0].replace(" ", "")
```

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

            if root_name == "":
                root_name = name

            values_dict = {"U": -1, "R": -1, "I": -1}
            values = []
            child_names = []
            for attribute in connection[1:]:
                if re.findall("^␣␣␣␣", attribute) or re.findall("^\t", attribute):
                    attribute = attribute.replace("␣", "").replace("\t", "")
                    values.append(attribute)
                else:
                    child_names.append(attribute)

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

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

        for elem_key in list(elements_dict):
            for child_name in elements_dict[elem_key][0]:
                if child_name in elements_dict:
                    elements_dict[elem_key][1].add_child(elements_dict[child_name][1])
                else:
                    elements_dict[elem_key][1].add_child(circuits.Element(child_name, "r"))

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

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


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

Listing 2: Sprache: Python

8.3 die Logik der Knoten

```
import schemdraw
import schemdraw.elements as elm
schemdraw.use('svg')


class Element:
    def __init__(self, name: str, type_str: str, resistance=-1.0, voltage=-1.0, current=-1.0):
        type_str = type_str.lower()
        types = {
            "r": 0,
            "s": 1,
            "p": 2
        }
        self.type = types[type_str]
        self.name = name

        self.resistance = float(resistance)
        self.voltage = float(voltage)
        self.current = float(current)

        self.child_Elements = []

        self.on_change()


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

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

        if self.voltage != -1 and self.resistance != -1 and self.current == -1:
            if self.resistance != 0:
                self.current = self.voltage / self.resistance
            else:
                print(f"Der Widerstand bei {self.name} ist 0.")
            return

        if self.voltage != -1 and self.current != -1 and self.resistance == -1:
            if self.current != 0:
                self.resistance = self.voltage / self.current
            else:
                print(f"Die Stromspannung bei {self.name} ist 0.")
            return

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

124         # get the amount of unknown values
            unknown_voltage = 0
126         unknown_current = 0
            unknown_resistance = 0
128         for child in self.child_Elements:
                if child.voltage == -1:
130                 unknown_voltage += 1
                    missing_voltage_elem = child
132             if child.current == -1:
                    unknown_current += 1
134                 missing_current_elem = child
                if child.resistance == -1:
136                 unknown_resistance += 1
                    missing_resistance_elem = child
138
            if self.type == 1:
140             # Reihenschaltung
                if unknown_resistance == 0:
142                 total_resistance = 0
                    for child in self.child_Elements:
144                     total_resistance += child.resistance
                    self.set_resistance(total_resistance)
146
                elif unknown_resistance == 1 and self.resistance != -1:
148                 total_resistance = 0
                    for child in self.child_Elements:
150                     if child.resistance != -1:
                            total_resistance += child.resistance
152                 missing_resistance_elem.set_resistance(self.resistance - total_resistance)

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

160             elif unknown_voltage == 1 and self.voltage != -1:
                    total_voltage = 0
162                 for child in self.child_Elements:
                        if child.voltage != -1:
164                         total_voltage += child.voltage
                    missing_voltage_elem.set_voltage(self.voltage - total_voltage)
166

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

```
176             elif unknown_current == 1 and self.current != -1:
                    total_current = 0
178                 for child in self.child_Elements:
                        if child.current != -1:
180                         total_current += child.current

182                 missing_current_elem.set_current(self.current - total_current)

184         self.maschen_knoten_self()
            self.maschen_knoten_children()
186
            for child in self.child_Elements:
188             child.compute()

190     def output_values(self, return_str=""):
            type_descriptions = ["Widerstand", "Reihenschaltung", "Parallelschaltung"]
192         return_str += f"{self.name}, {type_descriptions[self.type]}\n"
            print(f"{self.name}, {type_descriptions[self.type]}")
194         if self.resistance != -1:
                print(f"    R: {round(self.resistance, 3)}Ohm")
196             return_str += f"    R: {round(self.resistance, 3)}Ohm\n"
            if self.voltage != -1:
198             print(f"    U: {round(self.voltage, 3)}V")
                return_str += f"    U: {round(self.voltage, 3)}V\n"
200         if self.current != -1:
                print(f"    I: {round(self.current, 3)}A")
202             return_str += f"    I: {round(self.current, 3)}A\n"
            return_str += "\n"
204         print("")

206         if self.type == 1 or self.type == 2:
                for child in self.child_Elements:
208                 return_str = child.output_values(return_str=return_str)

210         return return_str

212     def add_child(self, child):
            self.child_Elements.append(child)
214         self.maschen_knoten_self()

216     def draw(self, d: schemdraw.Drawing, latest_elem):
            if self.type == 0:
218             d += (latest_elem := elm.Resistor().right().label(self.name).at(latest_elem.end))

220             return 1, latest_elem, 1

222         if self.type == 1:
                steps = 0
224             height = 1

226             for child in self.child_Elements:
                    r_steps, latest_elem, height = child.draw(d, latest_elem)
228                 steps += r_steps

230             return steps, latest_elem, height

232         if self.type == 2:
                steps, last, heights = self.child_Elements[0].draw(d, latest_elem)
234             parallel_steps = [steps]
                parallel_heights = [heights]
236             parallel_last = [last]

238             for child in self.child_Elements[1:]:
                    for n in range(parallel_heights[-1]):
240                     d += (latest_elem := elm.Line().up().at(latest_elem.end))
                    steps, last, height = child.draw(d, latest_elem)
242                 parallel_steps.append(steps)
                    parallel_last.append(last)
244                 parallel_heights.append(height)

246             # get max steps
                max_steps = 0
```

```
248             for step in parallel_steps:
                    if step > max_steps:
250                     max_steps = step

252             # draw rest of the circuit
                for i, latest_elem in enumerate(parallel_last):
254                 for n in range(max_steps - parallel_steps[i]):
                        d += (latest_elem := elm.Line().right().at(latest_elem.end))
256                 if i > 0:
                        for n in range(parallel_heights[i]):
258                         d += (latest_elem := elm.Line().down().at(latest_elem.end))
                    else:
260                     latest_elem_r = latest_elem

262             current_height = 0
                for heights in parallel_heights:
264                 current_height += heights

266             return max_steps, latest_elem_r, current_height

268     def draw_as_root(self, save_to=""):
            d = schemdraw.Drawing()
270
            d += (latest_elem := elm.SourceV().up().label(self.name))
272
            steps, latest_elem, height = self.draw(d, latest_elem)
274
            d += (latest_elem := elm.Line().down().at(latest_elem.end))
276         for i in range(steps):
                d += (latest_elem := elm.Line().left().at(latest_elem.end))
278
            d.draw()
280         if save_to == "":
                d.save('schematic.svg')
282         else:
                d.save(f"graphics/png/{save_to}.png", dpi=300)
284             d.save(f"graphics/{save_to}.svg")
```

Listing 3: Sprache: Python